```
<!DOCTYPE html>

<html>
<head>
    <meta charset="utf-8">
    <title>JS Play, v2_1</title>
    <style type="text/css">
        table { border: double; width: 60%; border-collapse: collapse }
        table > caption { margin-bottom: 10px; }
        th, td { padding: 15px; text-align: center; }
        /* adding some invisible/visible interaction rule(s) */
        .interaction { visibility: hidden; }
        /* improve table style? */
        th { background-color: lightgreen; }
        td {background-color: yellow; }
    </style>
</head>

<body>
<h1>Playing with JavaScript</h1>
<!--
The following table might just as well be
generated by the Script. But, because we already
know a lot about what it contains, it's easier to
specify its main features here and delegate the actual
computation and population of its cells to the script.
Pay attention to this: it's a common "pattern."
-->
<table id="interaction">
    <!--
    The "caption" element will be generated by the script because
    its contents depend upon what the user typed.
    -->
    <caption id="caption"></caption>
    <thead>
        <!--
        Because we already know what's in this table, it's okay
        to provide these details in the document, leaving the
        heavy lifting to the script.
        -->
        <tr>
            <th>Sum</th> <th>Absolute Difference</th> <th>Product</th> <th>Quotient</th>
        </tr>
    </thead>
   <!--
    As a matter of practice: whenever I use a "thead" I also use the "tbody"
    (and sometimes, if relevant) a "tfoot" element.
    -->
    <tbody>
        <tr>
            <!--
            Pay attention to the use of "ids" here.
            This allows the script to easily provide the correct
            results to the corresponding cell(s).
            -->
            <td id="sum"></td>
            <td id="difference"></td>
            <td id="product"></td>
            <td id="quotient"></td>
        </tr>
    </tbody>
</table>
<!--
Notice where I placed the Script in this buffer. Because the
script references objects in the DOM, I need to ensure that those
objects have "living" references when I attempt to dereference (use)
them in this script:
-->
<script type="text/javascript">
    /**
```

```
71          * Used by the various prompting routines to ensure that
72          * users do not exceed a range.
73          */
74         const MAX=100; // largest integer that user may enter.
75         const MIN=0; // smallest integer that the user may enter.
76         /* The following functions are used to simplify the "main" logic. */
77
78         /**
79          * Preconditions: the constants MIN and  MAX have been set.
80          * Postconditions: an integer >= MIN  but <= the MAX is returned.
81          * Note: this function will continue prompting the user until these conditions
82          * are met. This is an example of a "nag" function.
83          */
84         function promptInt() {
85             /*
86              * Ask the user once ... if all goes well, then the while statement that
87              * follows is never executed.
88              */
89             var input = window.prompt( "Enter an integer greater than or equal to " + MIN +  " but not greater than " + MAX + ":
     " );
90             /*
91              * Note: we distinguish between several "kinds" of iterative statements in programming
92              * languages.
93              * Bounded Iterators are constructions where the number of times that an
94              * iteration is performed is "known ahead of time."
95              * Unbounded Iterators: constructions where the number of iterations are unknown
96              * at the time the computation begins. Presumably, some condition becomes true or false
97              * and that signals the end of the iteration.
98              * In the usage below: we use the "while" statement which is an "unbounded" iteration construct.
99              */
100            while( input < MIN || input > MAX ) {
101                input = window.prompt( "Trying again: please enter an integer greater than or equal to " + MIN +  " but not grea
    ter than " + MAX + ": " );
102            }
103            /* if we ever get out of the unbounded while loop above, then we have a "safe" integer ... */
104            return parseInt( input );
105        }
106
107        /**
108         * Preconditions: Given two integers
109         * Postconditions: return the absolute difference between these two integers, meaning
110         * that the difference between these two integers as a non-negative integer is returned.
111         */
112        function absDifference( number1, number2 ) {
113            if( number1 < number2 ) {
114                return number2 - number1;
115            } else {
116                return number1 - number2;
117            }
118        }
119
120        /**
121         * Precondition: two integers are given, where the second is NOT zero.
122         * Postconditions: the "integer" quotient is returned, which is the floor of
123         * the actual quotient.
124         * [In this particular application, however, zeros should never appear.
125         * The conditional is used to demonstrate how the "alert" method might
126         * be used to help debug JavaScript ....]
127         * But, consider what might happen if we shared some of these functions
128         * with other web-pages ...
129         */
130        function intQuotient( number1, number2 ) {
131            if( number2 === 0 ) {
132                window.alert( "Attempted division by zero! ");
133                return 0;
134            } else {
135                return Math.floor( number1 / number2 );
136            }
137        }
138
139        /* end of private functions block */
140
```

```
141         /* Main logic: program execution starts here ... */
142
143         /**
144          * Note how the use of a function simplifies a tedious task here.
145          *
146          * [Think about how we might enhance this interaction in the future.]
147          */
148             var input1 = promptInt();
149             var input2 = promptInt();
150             /**
151              * Perform the computations. Think about the following: What are out "options"
152              * if the user entered something unexpected here? Suppose, for instance,
153              * that we wanted ONLY non-negative integers, but the user entered negative
154              * integers?
155              * Take a look at the Chapter on Forms (Chapter 7?) in your textbook and keep
156              * these kinds of questions in mind. Scripts are often used to "validate" forms
157              * input!
158              */
159             var sum = input1 + input2;
160             /** Note again: we use a function to "hide" any complicated processing. In
161              * essence, we define a new "verb" and use it.
162              */
163             var difference = absDifference( input1, input2 );
164             var product = input1 * input2;
165             /**
166              * The intQuotient does something unorthodox: if the
167              * second number is a zero, it complains and returns zero.
168              * BUT, this should NEVER happen ...
169              * Do you see why? (Hint: look at the promptInt function, above.)
170              */
171             var quotient = intQuotient( input1, input2 );
172             var remainder = input1 % input2;
173             /**
174              * Please observe the following "pattern." Get comfortable with it;
175              * you will use it throughout the remainder of the semester.
176              *
177              * Reflect on who "owns" these "elemens." Clearly, these are objects
178              * that reside in the "document."
179              *
180              * For those "forward thinking" readers: what would be the result of retrieving
181              * a reference to an HTML element that was associated with a "class" instead of
182              * an "id"? (Hint: what is the difference between things that are marked with
183              * "id"s and those marked with "class"es?)
184              */
185             document.getElementById( "sum" ).innerHTML=sum;
186             document.getElementById( "difference" ).innerHTML=difference;
187             document.getElementById( "product" ).innerHTML=product;
188             if( remainder === 0 ) {
189                 document.getElementById( "quotient" ).innerHTML=quotient;
190             } else { document.getElementById("quotient").innerHTML = quotient + ", with remainder: " + remainder; }
191             document.getElementById( "caption" ).innerHTML="Given " + input1 + " and " + input2 + ": computed the following...";

192             /**
193              * Turn on the visibility for objects of the class "interaction"
194              */
195             var interactionElements = document.getElementByClassName( "interaction" );
196             for( index=0; index < interactionElements.length; index++ ) {
197                 interactionElements[ index ].visibility=":visible";
198             }
199
200         </script>
201
202     </body>
203     </html>
```