Purpose:

This project is intended to give students experience in composing a JavaScript program that repeats steps and makes decisions. Unlike many simple programs that perform their steps in a linear sequence, this program will involve branching away from straight sequence and use more complex control structures: sequence and repetition. Although many different techniques exist for coding and executing scripts, this project will require that the script be coded and stored in an external file separate from the web page that uses it. The HTML file will be used to launch the script and as a window to receive the script's output. This program will use a "pop-up prompt (dialog) box" to receive its single item of user input from the keyboard and will produce HTML output in the web page using JavaScript `document.write()` methods. As with prior projects, all files in this project will be written using a plain text editor to help students develop a fundamental understanding of HTML and JavaScript syntax.

Program (Script) Objective:

Determine if a whole number, entered by the user at the keyboard, is a "prime number" (one that is evenly divisible by only itself or 1), and display a message indicating the results. Acquire the user's input via a dialog box similar to the one shown in the INPUT area of the Sample Output below. Display the results in the browser window in a manner similar to that shown in the OUTPUT area of the Sample Output below. Do not be concerned about the math in this problem. The steps to solve it are provided below. The only math you will need to understand besides simple counting is how to calculate a remainder using the modulus operator (%) as demonstrated on the W3Schools tutorial page about JavaScript Operators and within SkillPort.

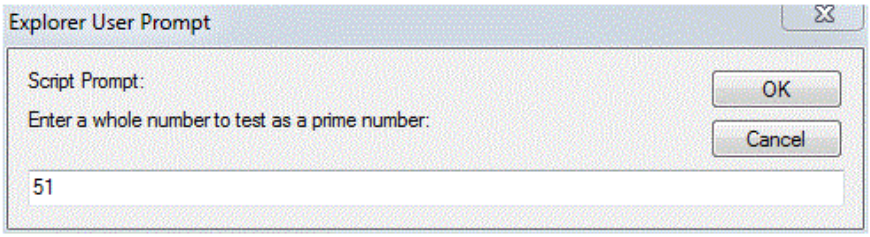
Sample Output:

The INPUT EXAMPLE illustration below shows a prompt dialog box that your JavaScript will have to produce to allow the user to enter the input value. The box can be executed via a method named `prompt()`. See the W3Schools tutorial page about JavaScript Popup Boxes. Note that the words "Script Prompt:" in the INPUT EXAMPLE illustration were produced by the browser used to produce the illustration. Those words or similar descriptive text will be produced automatically by whatever browser is used to run the script. They should not be part of your script instructions. Only the phrase "Enter a whole number to test as a prime number:" must be produced inside the prompt popup by your script.

The OUTPUT area of the illustration shows how your browser window should appear when the program is complete. The header will be produced as an h1 element in the body of the HTML file. The message explaining page's purpose will be produced as a paragraph element in the body of the HTML file. The message about the prime number status will be produced by statements in the JavaScript file.

NOTE: The final line shown in the OUTPUT EXAMPLE must be produced as a paragraph element in the body of the HTML file, not by your script.

INPUT EXAMPLE (via a Prompt Dialog Box):



OUTPUT EXAMPLE:

Variable List:

Your JavaScript code will declare its own variable labels to use to accomplish its task. The following "variable list" defines the labels you should use within your code.

LABEL	DESCRIPTION	Data Type	SOURCE	USAGE	DESTINATION
UI	User Input of number to test	String	Prompt dialog box	Parsed into an integer TV	Browser Document
TV	Test value acquired from user	Integer	Parsed from UI	Repeatedly divided by DD to test for prime number status	---
DD	Division denominator	Integer	Set to TV	Loop index and test divisor	---
HITS	Quantity of successful integer divisions	Integer	Set to 0	Conditionally incremented and tested	---

Script Algorithm:

You are not required to analyze the steps necessary to accomplish the objective of the script. That work has already been done for you, resulting in an "algorithm" (logical recipe) as follows.

A. Start.
B. Display a prompt dialog box and store the user's input in UI.
    i. Store UI acquired from a prompt dialog box.
    ii. Parse UI as a Base-10 integer and assign it to variable TV.
C. Initialize the HITS counter to zero.
D. Initialize the DD variable to the value of TV.
E. While DD is greater than zero, repeat the following block of two steps:
    i. Test if TV is evenly divisible by DD (that is, if the remainder of TV/DD equals zero).
    If so, increment HITS by 1.
    ii. Decrement DD by 1.
F. Display results within an HTML paragraph the browser's document window as seen in the Sample OUTPUT. The steps below must be performed in the order shown.
    i. Display the text stored in string UI.
    ii. Display the string " `is`"
    iii. If HITS is greater than 2, then display the string " `not`"
    iv. Display the string " `a prime number.`"
G. End.

Note that the steps above do not provide for all of the web page's content, as some of that will be produced by HTML elements (headers and paragraphs) as described in the Sample Output section above. Output steps B and F performed by the script should be handled by JavaScript code in the external file. Read the HTML *source code* for the example file entitled "Javascript in an external .js file" posted on my web site at:

http://www.gibsonr.com/classes/internet/examples/javascript/jsex_external.htm

Note that when you click on the link above, the script in that file will execute. You can press the Esc key to cancel the input and exit the script (or enter some test data). After the prompt popup disappears, you can right-click the background of the web page to view its page source code.

In order to execute the JavaScript content of that document, you might have to reconfigure your browser's security settings as described on the page about Browser Security Configuration for Scripts.

Web document content generated by script instructions in the body section (as shown in the example referenced above) will precede document content contained in the body section of the HTML file.

Project Constraints:

A. Use a plain text editor to create your HTML and JavaScript files. Name them "**m5a1.htm**" and "**m5a1.js**". You must not use Word, Wordpad, Expression Web, Dreamweaver, or any other web authoring software.

B. Do *not* use any images, color, etc., or other multimedia content on this project.

C. Place the call to the external JavaScript file (named m5a1.js) inside a JavaScript element within the body section of the HTML file. (See the source code for the JavaScript example mentioned above.)

D. Place appropriate comments at the beginning of the two source code files (HTML and JavaScript) identifying the: filename, yourself as the author, and noting the date.

E. Use the W3C Markup Validation Service to validate your HTML code

F. Use a JavaScript Validator such as the one at the JSHint Validation Service to validate your JavaScript code. Errors should be corrected. Warnings can be ignored.

Tips:

- Before attempting this assignment (graded assessment measure), be sure to have completed all Module 5 SkillPort training modules and readings, and review the COP 1830 Projects page for additional helpful tips. Also try the practice exercises related to Module 5 at: http://www.gibsonr.com/classes/cop1830/exercises/index.html
- Start working on your project as soon as possible. Do not wait until you believe that you have mastered all of the skills. If you encounter an obstacle that you cannot solve after checking SkillPort,

and other reference materials, send me a copy of the source code and request feedback. Use the Blackboard email system to send the message and attach your file(s) inside of a compressed archive (zipped folder) to prevent your message from being blocked by mail server virus filters. Do not simply paste a copy of your code into an email. That will not work. Remember that I will need time to respond. So give yourself that time by starting early.

- Build the project one step at a time. Try to create a simple web page with just one paragraph on it; then validate it. If it validates, then try adding some more elements and validate again, etc.
- Many systems are "case sensitive", so pay attention to upper and lowercase when typing URL's and filenames in tags. Save your work after each change before trying to view it in your browser. Also, you must press the RELOAD (a.k.a. REFRESH) button on the browser's button bar to see any changes made to a page since the previous rendering of it.
- The sample data and results show in the illustration at the top of this document are valid. So you can use them as text data to verify that your project is performing properly.
- While debugging your code, employ alert boxes to report when your script has successfully reached benchmarks in your code. (See the W3Schools tutorial page about JavaScript Popup Boxes.) Then use the //style of JavaScript comment notation at the front of those statements to "comment out" the alert statements prior to submitting your code. For example:

```
//alert ("Input section of script reached.");
```

- If you are authoring using Windows software, you will find it easier to locate and select files on your computer if your Windows "Folder Options" view properties are set so that the software does not hide the extensions such as ".html" at the ends of filenames. Instructions for unhiding extensions can be found at the bottom of the web page about Windows® File Association.
- Save your work after each change before trying to view it in your browser. Also, you must instruct your browser to reload your page after saving any changes to it. Simply pressing the RELOAD (a.k.a. REFRESH) button on the browser's button bar many not be enough to make all browsers completely reload the revised script. To ensure that the page is completely reloaded, position your typing cursor at the end of the address of your web page file on the browser's address bar and press the Enter key on your keyboard. Be sure that you have deselected the address on the address bar before pressing Enter to avoid clearing the address with that keystroke. One way to do this is to press the End key on your keyboard after clicking in the address bar area.
- A remember, relax! Don't get stressed about this. Take it in small steps. Get as much of it working as you can. If you get stuck on one element, then use comment tags (<!-- and -->) to mask any code that is causing trouble, move on to a different element, and come back to solve the problem later. Some frustration can be a valuable motivator to reinforce lessons. But too much can inhibit learning. Schedule your time to allow some breaks between working sessions.

Submission:

Copy your **M5A1** folder into a compressed folder with the same name, but with a ".zip" extension. In Windows, this is performed by right-clicking the folder to be compressed and then selecting: Send to > Compressed (zipped) folder .

Your **M5A1.zip** file should be submitted via the **M5A1** Assignment page found in Blackboard under Content > Module 5 > Module 5 Assessments.

If you need step-by-step directions, read the web page at:
http://www.gibsonr.com/classes/howto/onlinesubm.html