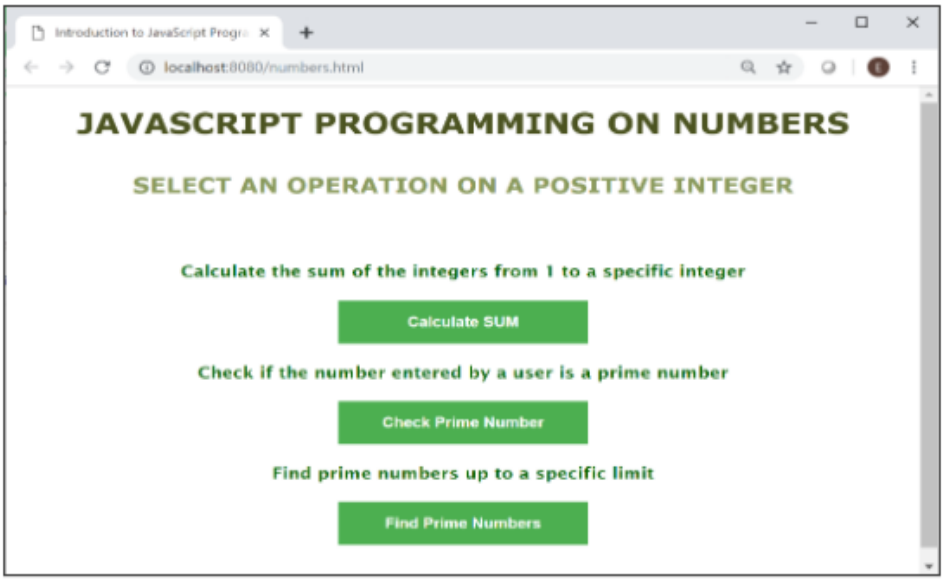
Lab 8 – JavaScript Programming

Overview

In this lab, you will implement three mathematic operations on positive integer numbers:

- Calculate the sum of the integers from 1 to a specific integer
- Check if an integer is a prime number
- Find all prime numbers up to a specific limit

To program your webpage, you will use JavaScript input, output, arithmetic, conditional, and loop statements using numbers and arrays to implement these operations.



Part 1 - Calculating the Sum of Numbers from 1 to a Specific Number

Part 1-1 - Creating a Function (10 Points)

If you want to calculate the sum of the numbers 1 to a specific number, the same block of code (that you wrote for Exercise 8) can be repeatedly reused for any given number. A function is a block of statements designed to perform a specific task, which can be used repeatedly in a program. You can define your own function once and use it many times.

A JavaScript function is defined with the `function` keyword, followed by a name, followed by parentheses (). Function parameters are listed inside the parentheses () in the function definition. A function has zero or more input parameters. Inside the function, the parameters behave like local variables. The code executed by the function is placed in curly brackets {}. A function can return a value. The keyword `return` ends the function and return a value. Input parameters send values to your function and a return statement returns values from your function.

```
function name(parameter1, parameter2, parmeter3) {
    // code to be executed
    return value;
}
```

Once a function is defined, you can then invoke it by calling that function in JavaScript code.

On your own, create a function to add the numbers 1 to a number entered by the user, call the function, and display the result on the HTML page, an alert dialog, and a browser console.

1. Create an HTML file named `numbers_function.html`. Add the usual HTML elements. If you did Exercise 8, you can save `numbers_simple.html` as a new file and modify it.
2. In the <script> element, define a function named "calculateSum" that performs the following:
   - Has one input parameter, which is a positive integer
   - Calculate the sum of the numbers 1 to the input positive integer
   - Has one output parameter, that returns the sum
3. You can call the function by passing an input value (e.g. 100). You can use a variable to store the return value from the function (just like Python!):

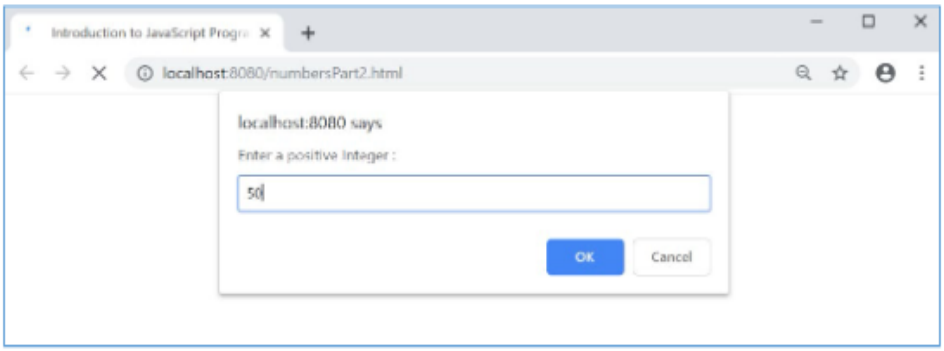
```
var result = calculateSum(100);
```

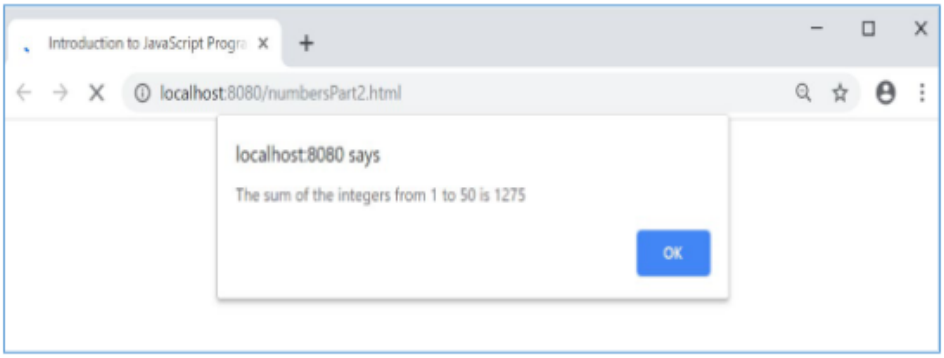
4. Display the result: (1) on the HTML page, (2) in an alert popup window, and (3) on the browser console (just like we did in Exercise 8). Save and preview the page.

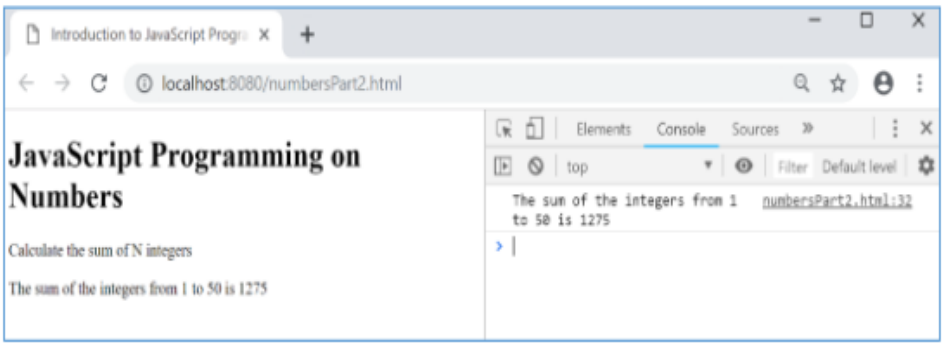
Part 1-2 - Prompt the User for Custom Input (10 Points)

The argument value of the function is hard-coded in Step 3 above. Let's change it. Instead, ask the user to enter a positive integer and call the function by passing the user-specified integer.

- Prompt the user for a positive integer using a prompt dialog. The `window.prompt()` function displays a dialog with an optional message prompting the user to input text.
- Convert the input text to an integer using the `parseInt()` function.
- Call the `calculateSum` function by passing the user-defined integer
  (Make sure you delete `calculateSum(100)`... There is no need for that code now.)
- Get the return value and assign it the result to a variable in your script.
- Display the result: (1) on the HTML page, (2) in an alert popup window, and (3) on the browser console. Save and preview the page.







Part 1-3 - Event Handling (10 Points)

A web browser will execute JavaScript code in the order it finds them in <script> elements and carry out the code when events occur. An HTML event can be something the browser does, or something a user does. There are many different types of events that can occur, for example:

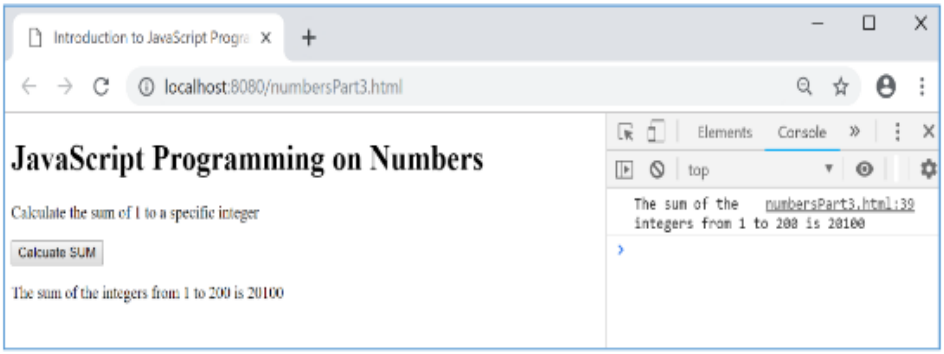
- An HTML web page has finished loading.
- An HTML input field was changed.
- An HTML button was clicked.

When events happen, you may want to do something. HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

Now, invoke the `calculateSum()` function when an HTML button is clicked.

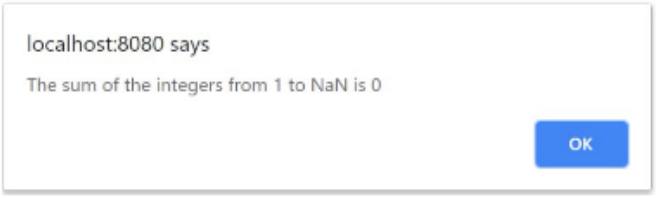
1. Modify the file `numbers_function.html`. The tasks for Part 1-3 are to: (1) move the code you wrote for Part 1-2 into a function and (2) execute the function with a button click.
2. In the <script> element, create a function named "executeOperationsOnNumbers" that will be called when an HTML button is clicked. This function should perform the following:
   - Has no input parameters (at this point).
   - Prompts the user to enter a positive integer using a prompt dialog.
   - Convert the input text to the integer using `parseInt()` function.
   - Call the function `calculateSum()` by passing the integer and get the return value.
   - Display the returned value on the page, in an alert popup, and on the browser console.
3. Add a clickable <button> element to the <body> element. Put the text content to explain what the button does between <button> and </button> tags.
4. When the button is clicked, invoke `executeOperationsOnNumbers()` function by adding an `onclick` attribute to the <button> element.

5. Save and preview the page. Click the button to test your script.

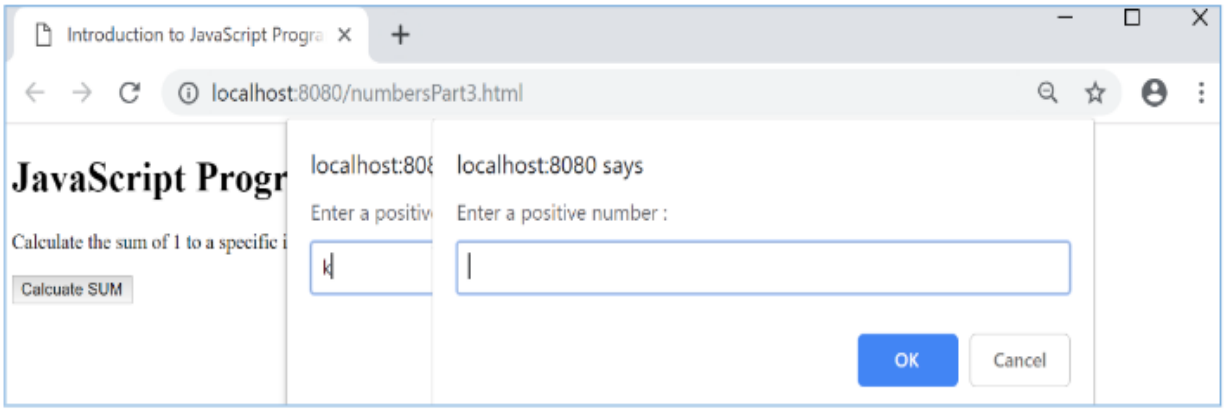


Part 1-4 - Input Validation (10 Points)

6. If you enter a non-numeric or negative value, it will show the following, which is incorrect.



7. Modify your script to perform input validation. Have your script continue to ask the user for a positive number until they enter a valid number:
   - Use an infinite `while` loop (e.g. while(true), while(1), etc.)
   - In the while loop:
     - Prompt the user to enter a positive integer.
     - Parse the string input to an integer using `parseInt()` function.
     - You can check to see if the parsed integer is an integer or NaN (not a number) by using the `isInteger()` method of the `Number` object.
     - Break out of the loop only if the input **is an integer** and **greater than zero**.
     - Otherwise, your script should loop and prompt the user again.
8. Save and preview the page in a browser. Click the button and test your script.



Part 2 - Creating a Website to Perform Different Operations on Numbers

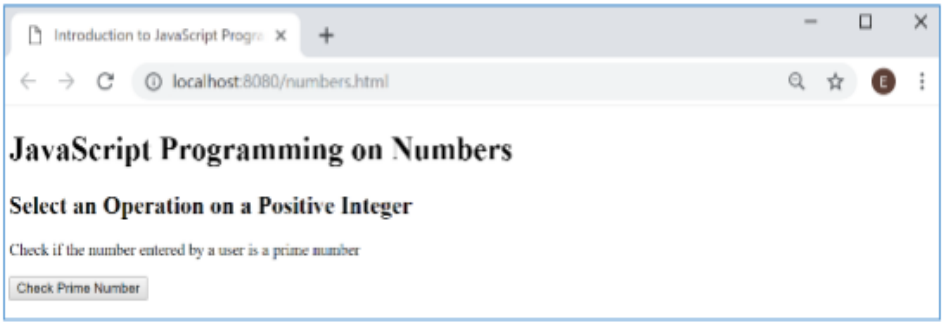
Part 2-1 - Checking a Prime Number (10 Points)

Now you will create an external JavaScript file and use it in an HTML file. In the external JavaScript file, you will create a function that checks if the user-specified integer is a prime number. A prime number is a positive integer that is divisible only by only 1 and itself. For example: 2, 3, 5, 7, 11, 13, etc. Once you define the function, call the function by passing a user-entered integer and display the result on the HTML page, an alert dialog, and a browser console.

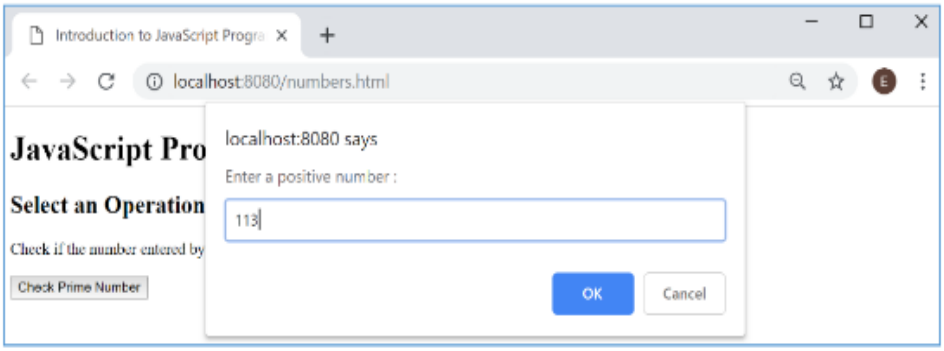
1. Create an HTML file named `numbers.html`.
2. Write <!DOCTYPE>, <html>, <head> and <body> elements in `numbers.html`.
3. Add <meta> tag to the head section to specify character encoding.
4. Insert a title (e.g. Introduction to JavaScript Programming) to the head section of the page.
5. Create an external JavaScript file named `numbers.js`. The name of the file is up to you.
6. Place a <script> element in either the <body> or the <head> element.
7. To use an external script, put the name of the script file in the src attribute of a <script> tag.

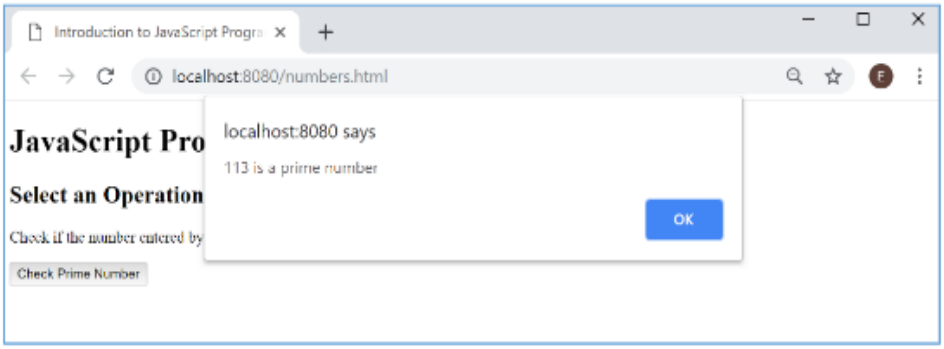
8. In the script file, define a function named "checkPrimeNumber" that performs the following:
   - Has one input parameter (e.g. n), which is assumed to be a positive integer.
   - Declare a variable (e.g. isPrime) storing true or false and assign `true` initially.
   - If the input is equal to 1, set the flag variable, isPrime to `false`, since 1 is not prime.
   - Otherwise, repeat the following steps from `i = 2` until `i <= n/2`:
     - If the remainder of `n` divided by `i` equals to 0, it means the number is not prime:
       - Set `isPrime` to `false`.
       - Break the loop statement.
     - Otherwise, your loop should keep looping and iterating i.
     - Declare your variable i with `let` so it's local to this block of code! Otherwise if you use the same variable in other functions they will share the same variable!
   - The function should have one output parameter that returns the variable, `isPrime`.

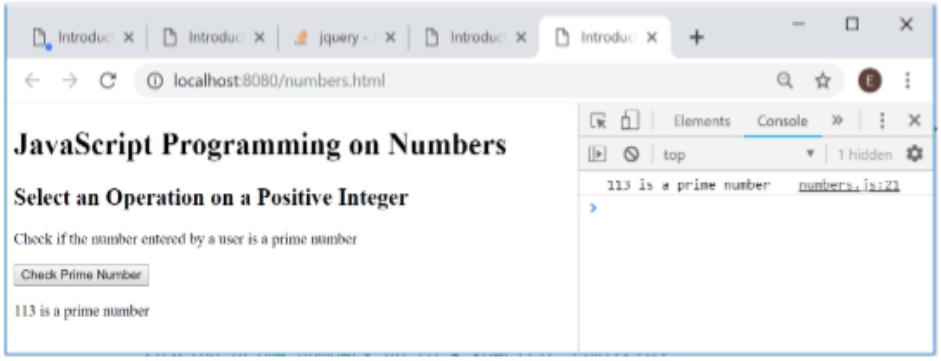
9. Add the following elements to the <body> element (see example screenshot below).
   - Two headings describing the page.
   - A paragraph describing the operation the "checkPrimeNumber" function performs.
   - A clickable button.
   - A paragraph displaying the result of the operation.



10. In your script file, create a function named "executeOperationsOnNumbers" that will be called when an HTML button is clicked. This function is very similar to what you created in Part 1 and should perform the following:
    - Has no input parameters (at this point).
    - Prompt the user to enter a positive integer until they type in the valid numeric value.
        - In the infinite `while` loop:
            - Prompt the user to enter a positive integer.
            - Parse the string input to the number using `parseInt()` function.
            - Break the loop if the parsed number is an integer **and** greater than zero.
    - Call the function "checkPrimeNumber", pass the input integer, and get the return value.
    - Display a message saying if the input integer is or is not a prime number: (1) on the HTML page, (2) in an alert popup window, and (3) on the browser console.
        - Hint: Use an `if` statement to determine which message to display (is or is not).
11. When the button is clicked, invoke the executeOperationsOnNumbers() function by adding an `onclick` attribute to the <button> element.
12. Save and preview `numbers.html` in a browser.





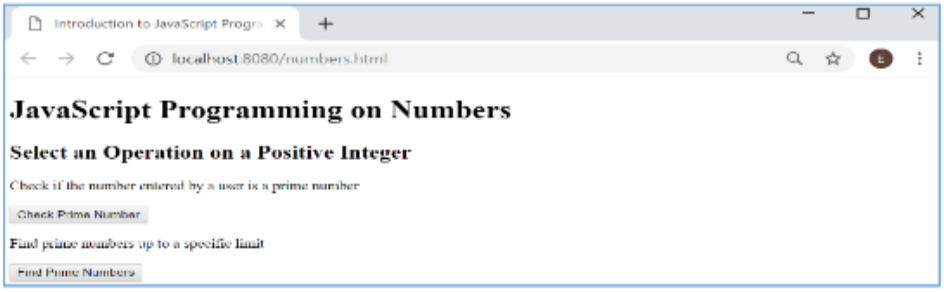


Part 2-2 - Finding Prime Numbers (10 Points)

You will continue to work on `numbers.html` and `numbers.js`. You will create a function to find all prime numbers up to the user-entered number. Once you defined the function, you will invoke the function by passing the input number when an HTML button is clicked, and display the result on the HTML page directly, in an alert dialog, and in the browser console.

1. Open `numbers.js`.
2. In the script file, create a function named "findPrimeNumbers" performing the following:
   - Has one input parameter (e.g. n), which is assumed to be a positive integer.
   - Declare an empty array. https://www.w3schools.com/js/js_arrays.asp
   - Repeat the following steps from `i = 2` for `i <= n` (`n` is the input number):
     - Call `checkPrimeNumber()` to see if the current number `i` is a prime number.
     - If `i` is a prime number, add it to the array.
     - Declare your variable i with `let` so it's local to this block of code! Otherwise if you use the same variable in other functions they will share the same variable!
   - Return the array containing the prime numbers from 1 to `n`.

3. Open `numbers.html`. Add the following elements to the <body> element:
   - A paragraph describing the operation the "findPrimeNumbers" function performs.
   - A clickable button for the "findPrimeNumbers" (like you did for "checkPrimeNumber").



Part 2-3 - More on Event Handling (10 Points)

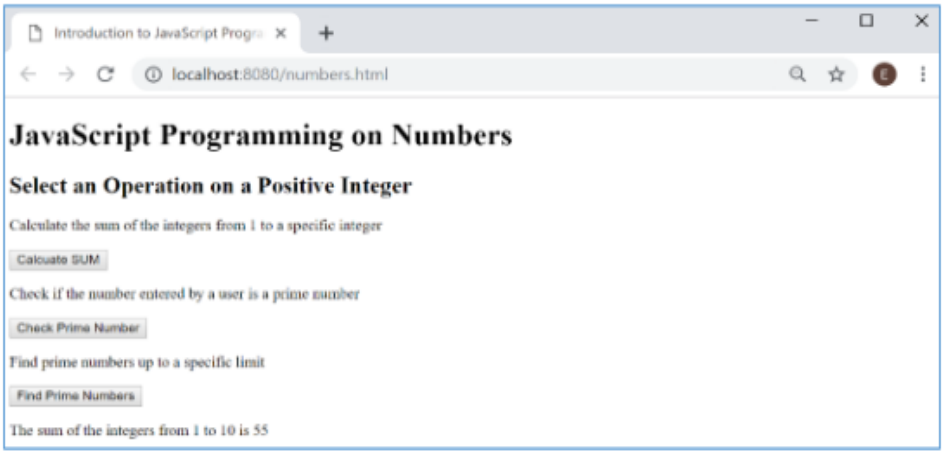
4. Now you will modify the function "executeOperationsOnNumbers" to handle all click events that may occur on any of the buttons on your page. This function performs the following:
   - Has one input parameter indicating the operation (i.e. function) that should be performed.
     - The name and data type of the parameter is up to you.
     - Let's say this function has a parameter named `op`.
       - It could store strings: for example, "checkprime" and "findprimes" to indicate each operation, respectively.
       - It can store integer values: 1 and 2 to indicate each operation. For example, 1 for "checkprimeNumber" and 2 for "findPrimeNumbers"
     - The code that I describe below will use a string to indicate each operation.
   - Prompt the user to enter a positive integer until they type in the valid numeric value.
     - In the infinite `while` loop:
       - Prompt the user to enter a positive integer.
       - Parse the string input to the number using `parseInt()` function.
       - Break the loop if the parsed number is an integer **and** greater than zero.

- Use a `switch` statement to perform different blocks of code depending on the value of the input argument, `op`:
    - When the value of `op` is equal to "`checkprime`", call `checkPrimeNumber()` by passing the user-entered number and display the result on the page, in an alert dialog, and on a browser console.
    - When the value of `op` is equal to "`findprimes`", call `findPrimeNumbers()` by passing the user-entered number. Display the prime numbers and the count of the prime numbers on the page, in an alert dialog, and on a browser console.
    - If there is no match, the default block of code is executed, which should alert the user that no known operations was found.
    - It is necessary to break the case blocks except for the last one, which is usually the default case because the program will continue to execute the next block until it meets a break. The last one ends there anyway.

```
switch(op) {
      // The case of "checkprime"
            // Run the "checkPrimeNumber" function
            break;
      // The case of case "findprimes"
            // Run the "findPrimeNumber" function
            break;
      // The default case
            // Alert user that no known operation was found!
}
```

5. If "Check Prime Number" is clicked, run `executeOperationsOnNumbers('checkprime')`. The argument value "`checkprime`" must match one of the cases in the switch statement.
6. If "Find Prime Number" is clicked, run `executeOperationsOnNumbers('findprimes')`. The argument value "`findprimes`" must match one of the cases in the switch statement.
7. Save `numbers.html` and `numbers.js`, preview `numbers.html` in a browser.

8. Open `numbers.html`. Add a button and a paragraph for an operation that adds 1 to a user-defined number (n) (i.e. the "calculateSum") function which is implemented in Part 1).
    - Note: You need to add the function `calculateSum` implemented in Part 1.
9. Modify the script written in `numbers.js`. When the "Calculate Sum" button is clicked, call `calculateSum()` by passing a user-specific number and display the result on the HTML page, in an alert popup window, and on a browser console.
10. Save and preview `numbers.html`. Your page should look like the following figure.



Part 2-4 - More on Input Validation (5 Points)
Question - When a prompt dialog asks for a positive integer value, you may click the "cancel" button to stop executing the program. What happened when you clicked the "cancel" button? It will continue to ask the user to enter a positive integer and never close the prompt dialog.

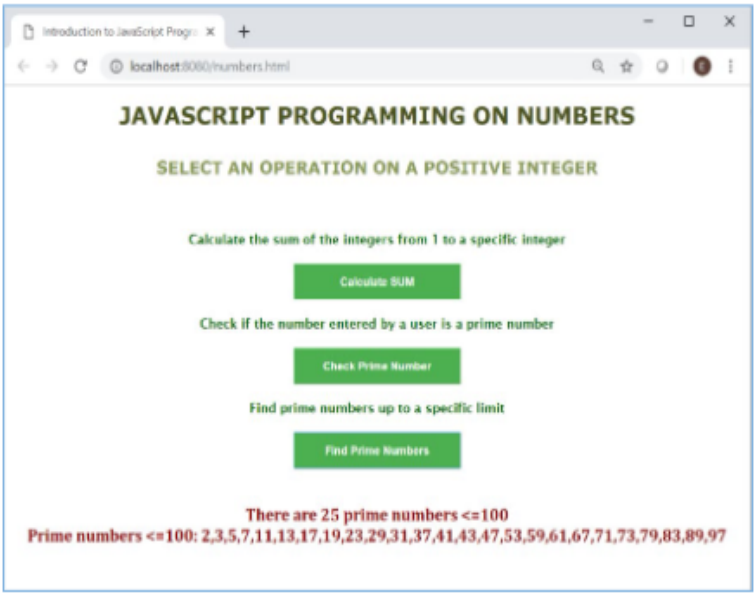
How can you terminate the program and close the prompt when you click 'cancel' button?

On your own, open `numbers.js` and modify your script to solve this question.

Part 3 - Adding Styles to your HTML page (10 Points)
Now you should apply styles to the page `numbers.html` you implemented in Part 2.

1. Create a stylesheet named `numbers-style.css`. The name of the CSS file is up to you.
2. Specify the external stylesheet file (`numbers-style.css`) in the head section of `numbers.html` using <link> tag.
3. Your stylesheet should do the following:
    - Center the headings, paragraph texts, and buttons horizontally.
    - Change the text color of the headings and paragraphs.
    - Change font-family, font-weight, and font-style of headings, paragraphs, and buttons.
    - Set the width of the buttons.
    - Set the background-color of the buttons.
    - Add space between the elements by setting margin and padding properties.

Note: You can be as creative as you like. The following is an example of the page with CSS.



Part 4 - Verifying the HTML and CSS of your Web Site (5 Points)

Before you submit Lab Assignment 8, check errors in your HTML and CSS files using:

- http://validator.w3.org (HTML5)
- https://jigsaw.w3.org/css-validator/ (CSS3)

Any significant errors caught by the validators will result in points taken off.

Part 5 - Adding Comments (5 Points)

You must include appropriate comments in your JavaScript code for full credit. For example:

```
/* function name: calculateSum
* Purpose: Add the numers from 1 to n
* Inputs: n - a positive integer
* Returns: sum - the sum of the numbers 1 to n
*/
function calculateSum(n) {
     sum = 0;  // assign 0 to sum initially
     //code executed …
     return sum;
}
```

Part 6 - Publishing a Website on TerpConnect (5 Points)

1. Upload all the HTML, CSS, and JS files and publish your website for Lab 8 on TerpConnect.
2. View your web pages in a browser and verify you did the lab assignment successfully.
   e.g. http://www.terpconnect.umd.edu/~resop/geog646/lab8/numbers.html

What to Submit

Submit a compressed (ZIP) file of your lab directory containing HTML, CSS, and image files on ELMS → Assignments → Lab 8. Leave a comment with the URL of your web page.