

# EECS 1012. LAB 4: HTML + JS + Computational Thinking (Oct 5–9, 2020)

## A. IMPORTANT REMINDERS

- 1) Each lab including the pre-lab mini quiz is about 2.0 % of your overall grade.
- 2) You must attend your own lab session (the one you are enrolled in). If you need to change your lab enrollment, you should go to the department. Instructors or TAs cannot change your enrollment. TAs are available via Zoom to help you, and you are welcome to attend these online lab sessions (the attendance is optional, but is highly recommended). You can also have your work verified and graded during the lab sessions. Feel free to signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too. In case you run out of time, the submission you make over eClass will be marked by the TAs after the lab ends (possibly not by the same TAs who assisted you during the lab session).
- 3) You can submit your lab work anytime before the deadline. We do not accept late submissions.
- 4) You must complete the pre-lab mini quiz posted on eClass no later than the first 15 minutes of your lab time.

## B. IMPORTANT PRE-LAB TASKS YOU NEED TO PERFORM BEFORE THE LAB

- 1) Download this lab files and review them completely.
- 2) You should have a good understanding of
  - Events (such as onclick) and event handlers
  - `document.getElementById().innerHTML`
  - JavaScript functions such as `parseInt()`, `parseFloat()`, `toFixed()`
  - the `if` statement and `switch` statement in JavaScript
  - memory space (aka variable), JavaScript operators such as “+” and the concept of overloading
  - flowchart symbols as described in the lecture notes
- 3) Practice drawing flowchart symbols in `draw.io`, MS Word, or PowerPoint. If you plan to draw your flowcharts on paper, please make sure you have pencils, erasers, and perhaps a ruler.

## C. GOALS/OUTCOMES FOR LAB

- To practice *computational thinking* by first drawing flowcharts for basic computational problems, followed by their implementation in JavaScript.

## D. TASKS

- 1) You first and major task in this lab is to draw eight flowcharts. This task should preferably be done in teams of two (not more). You may work solo in case you are unable to find a partner for this part. These drawings will need to be submitted as image files. Only **png** or **jpg** formats are acceptable.
  - Note you should NOT open VS-Code or browsers before finishing this task. You may draw your flowcharts on paper, or using `draw.io`, MS Word or PowerPoint.

- 2) Then, you are provided with `ct.html` document and with supporting files such as `ct.css` and `ct.js`. Your task is to translate your *first five* flowcharts to a JavaScript code.
- 3) You will generate at least five `html` and `js` files in this process. If you plan on demoing the files to the TAs, please have each `html` file open in a different tab to facilitate the process.
- 4) See the following few pages for details on how to modify your `html` and `js` files.

## E. SUBMISSION

### 1) Manual verification by a TA (optional)

You may have one of the TAs verify your lab before submission. The TA will look at your various files in their progression. The TA may also ask you to make minor modifications to the lab to demonstrate your knowledge of the materials. The TA can then record your grade in the system.

### 2) eClass submission

You will see an assignment submission link on eClass. Create a **folder** named “**Lab04**” and copy all of your lab materials inside (`img_{01,02,03,04,05,06,07,08}.jpg`; `ct_Ex{1,2,3,4,5}.html` and `ct_Ex{1,2,3,4,5}.js`). This folder should be compressed (**zip** or **tar.gz**) and the compressed file submitted.

In case you work with a partner – who must be from the same lab section, **both** people should submit the files. However, an extra file, **group.txt**, should be included in the submission, containing the full names and the student numbers of the team members. Both students will then receive the same grade. Note that the pre-lab quizzes are still expected to be done by each student separately.

## Part 1

Preferably in teams of two. If you have already [mostly] completed it, you must discuss it with your peer before you show your final solution to your TA or submit it.

Using a computer program (or a pen and pencil), draw the following flowcharts and write your name on each. You are required to use the symbols introduced in the slides, for all eight exercises. At the end, you should take a screenshot (or a picture) of each flowchart and submit them to eClass as **img\_{01,02,03,04,05,06,07,08}.jpg** files, where **img\_x** is the flowchart of exercise **x** below.

In case you submit images produced by a phone camera, make sure you change the file format in the camera settings to JPG (as opposed to HEIF, DNG, etc.) Feel free to reduce the image dimensions to reduce the file size – either in the camera settings or in an image editor. Try to keep the size of each image below 500 KB. Also, using *exposure correction* of +0.6 or +1 in the camera makes the images look better (you can also brighten them afterwards in an editor).

**IMPORTANT:** You will be expected to perform a similar task (creating/drawing flowcharts and submitting them as images) during some of the tests; thus, make sure you master the procedure.

- Ex 1) draw a flowchart for a computer program to receive two numbers as sides of a rectangle and output the rectangle's perimeter.
- Ex 2) draw a flowchart for a computer program to receive three numbers and store them in memory spaces called *a*, *b*, and *c* as three semi-axes of an ellipsoid, and calculate and output the volume of this ellipsoid<sup>1</sup>.
- Ex 3) draw a flowchart for a computer program to receive three numerical coefficients of a quadratic equation (store them in memory spaces called *a*, *b*, and *c*) and calculate and output its roots. (assume coefficients are good enough such that a solution in real number exists. Don't worry about cases that a solution does not exist). If you need to refresh your memory on this topic, this might be a good source:  
<https://www.mathsisfun.com/algebra/quadratic-equation.html>
- Ex 4) draw a flowchart to receive three numerical coefficients of a quadratic equation and determine if it has two distinct real roots, one root, or no roots in real numbers. This page might be a good reference: <https://www.math10.com/en/algebra/quadratic-equation.html>
- Ex 5) draw a flowchart to receive a number and map it to a letter grade based on York standard. You may need to look at this reference: <http://calendars.registrar.yorku.ca/2012-2013/academic/grades/index.htm> Assume if the grade is between 40 and 49, it's mapped to an E.
- Ex 6) assume there is a webpage containing an HTML input of type text and a button. When the button is clicked a function, named *Problem06*, is called. Draw a flowchart that outputs whether the input is positive or negative until a zero is received. When a zero is received, the button is disabled (so the function cannot be called anymore).
- Ex 7) By modifying your flowchart above, draw a flowchart to continue receiving numbers and output if they are positive or negative, until a zero is entered. When a zero is entered, the

---

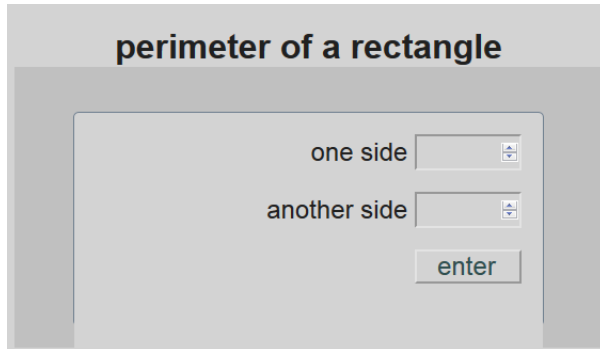
<sup>1</sup> <https://www.google.ca/search?q=ellipsoid+volume>

program should output how many positive and how many negative numbers have been entered, and then stop.

- Ex 8) Considering the same approach above, draw a flowchart to continue receiving numbers and output if they are divisible by 6 or not until a zero is entered. When a zero is entered, the program should output how many of the entered numbers were divisible by 6, then it stops. IMPORTANT RESTRICTION: you are not allowed to divide the number by 6; therefore, you are not allowed to use the remainder operator (%) over 6 to verify through the remainder if the number is divisible by 6. You may use any other math trick you wish.

## Part 2

You are given **ct.html**, **ct.css**, **ct.js** files. Review these files carefully, trying to understand the purpose of each line.



The screenshot shows a web form with a title "perimeter of a rectangle". Inside the form, there are two input fields. The first is labeled "one side" and the second is labeled "another side". Below these fields is a button labeled "enter".

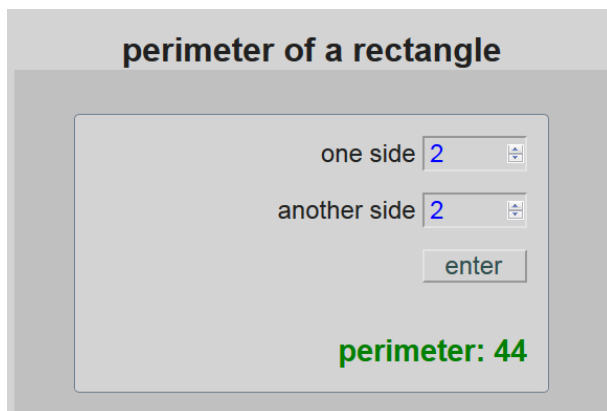
**Exercise 1.** Copy **ct.html** to a new file named **ct\_Ex1.html**. Copy **ct.js** to a new file named **ct\_Ex1.js**.

Launch **ct\_Ex1.html** with your browser and enter two numbers and click on the “enter” button, nothing happens. Let’s fix it:

Make three changes to **ct\_Ex1.html**, as follows:

- 1) Connect it to **ct\_Ex1.js** by adding a link in the head element.
- 2) Add an event to the button such that when it’s clicked you handle that event by the **perimeter()** function in your **js** file.
- 3) Add your name to the list of authors of this page.

Launch **ct\_Ex1.html** with your browser and enter 2 and 2. You should see the following result:



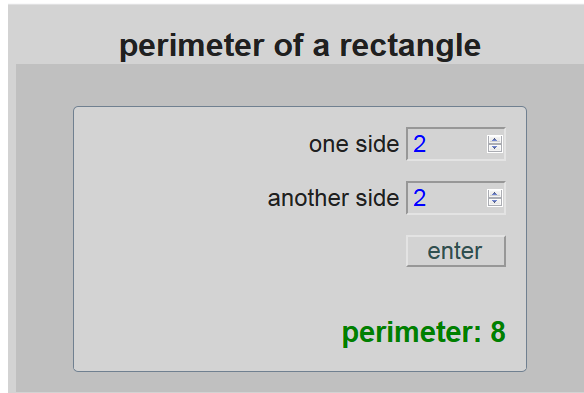
The screenshot shows the same web form as before, but now the input fields "one side" and "another side" contain the number "2". Below the "enter" button, the text "perimeter: 44" is displayed in green.

That is because the variables **w** and **h** in your **js** function are from data type “string”. The data type variables in **js** are determined by the data type of the expression that is on the right side of the assignment. In this case, because the data type of property value of the **html** object that the **getElementById** returns is “string”, therefore the data type of **w** (and **h**) is a “string” too.

Also, the “+” operator is overloaded in **js** (and many other programming languages), that means “+” has more than one meaning in **js**. As far as our concern is, one meaning is to concatenate two strings, and another meaning is to add two numbers. Because **w** and **h** are currently strings, “+” concatenates them, and the result is “22”, which is then converted to a number 22 to be multiplied (as **\*** makes no

sense when applied to strings, but can work if that string is interpreted as a number). `parseInt()` is a function that receives a string as its argument and returns its equivalent as an integer number. In other words, it can receive “15” and return 15. It can also receive “9” and return 9.

In summary, “2” + “2” results in “22”, instead of 4. So, open your **ct\_Ex1.js** file and by using `parseInt`, modify the code such that sum of the two inputs is calculated, not their concatenations.

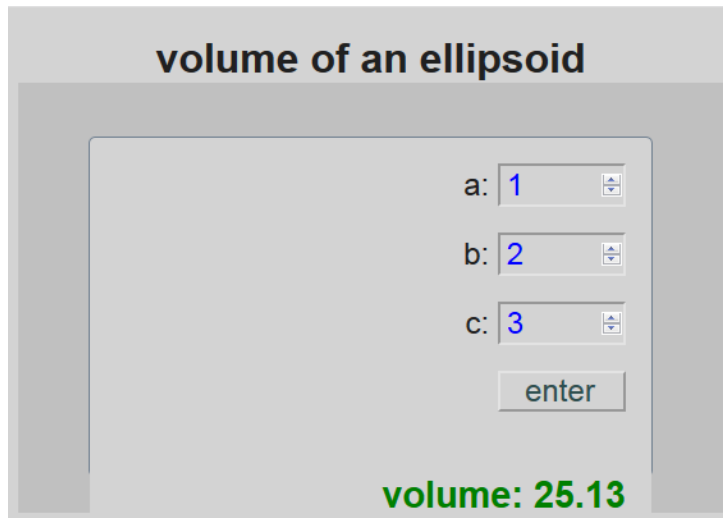


The image shows a web form with a title "perimeter of a rectangle". Inside the form, there are two input fields. The first is labeled "one side" and contains the number "2". The second is labeled "another side" and also contains the number "2". Below these fields is a button labeled "enter". At the bottom of the form, the text "perimeter: 8" is displayed in green.

Before moving on to Exercise 2, compare the code in **ct\_Ex1.js** with your **img\_01.jpg**, the flowchart you drew in Exercise 1 of Part 1. Are they conceptually the same? If the answer is “no”, something is wrong. Either modify your flowchart to be a good match to this code or provide a `js` code which is equivalent to your flowchart. However, note that in the flowchart, we do not go to details of languages. For instance, we assume “+” means *addition* as it does in real world; but, if in a programming language “+” has been overloaded, that’s the responsibility of the programmer (not the designer of the flowchart) to use it properly. As another example, the designer—in their flowchart—does not get involved in how `w` and `h` should be inputted. The designer just states to input `w` and `h`. It’s the task of the programmer to translate the flowchart to an appropriate statement in the target programming language, here JavaScript; elsewhere, Java, Python, C, etc. This is true for everything else in the design. One advantage of drawing a flowchart first is that you focus on the design, the process of computational thinking, instead of getting distracted by errands of the programming language, such as how to input, how to convert from string to number, what the right syntax is, etc. This becomes very critical when you want to tackle bigger projects.

**Exercise 2.** Copy **ct\_Ex1.html** and **ct\_Ex1.js** to new files named **ct\_Ex2.html** and **ct\_Ex2.js**.

In this exercise, you should translate your flowchart of Exercise 2 of Part 1 to js. You should make some changes in both html and js files, such that you get the following results when, for instance, you enter 10, 11, and 4 for the semi-axes of an ellipsoid.



The screenshot shows a web interface for calculating the volume of an ellipsoid. The title is "volume of an ellipsoid". There are three input fields labeled "a:", "b:", and "c:" with values 1, 2, and 3 respectively. Below the inputs is an "enter" button. At the bottom, the text "volume: 25.13" is displayed in green.

In particular, you need to make about 6 changes in your **ct\_Ex2.html** as follows:

- Make sure you link your html file to your new js file
- Change the header to be "volume of an ellipsoid"
- Add a new input (because this program receives 3 inputs) with id "num3"
- Correct the labels of the inputs to a, b, and c
- As semi-axes of an ellipsoid cannot be negative numbers change the min and max to 1 and 100
- When the "enter" button is clicked, the event handler `volume()` should be triggered.

Also, you need to make about 4 changes in your **ct\_Ex2.js** as follows:

- Make sure name of the function is `volume()`
- Add a line to capture the value of c
- Replace the line `p = 2 * (w + h);` with what you have in your flowchart
- Correct the output to have a more relevant label and correct value as the picture above shows

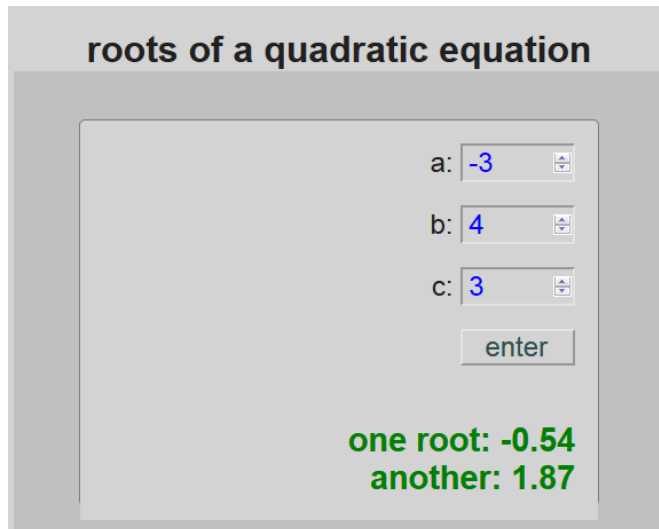
Note that the answer is fixed to 2 digits after the decimal point. This should be addressed in the implementation, not necessarily the design.

Before moving on to Exercise 3, compare your code in **ct\_Ex2.js** with your **img\_02.jpg**, your flowchart for this problem. Make sure they match.



**Exercise 3.** Copy **ct\_Ex2.html** and **ct\_Ex2.js** to new files named **ct\_Ex3.html** and **ct\_Ex3.js**.

In this exercise, you should translate your flowchart of Exercise 3 of Part 1 to js. You should make some changes in both html and js files, such that you get the following results when, for instance, you enter -3, 4, and 3 as the coefficients of the quadratic equation.



**roots of a quadratic equation**

a:

b:

c:

one root: -0.54  
another: 1.87

In particular, you need to make about 4 changes in your **ct\_Ex3.html** as follows:

- Make sure you link your html file to your new js file
- Change the header to be “roots of a quadratic equation”
- Change back min and max to -32768 and 32767, respectively
- When the “enter” button is clicked, the event handler `equation()` should be triggered.

Also, you need to make about 3 changes in your **ct\_Ex3.js** as follows:

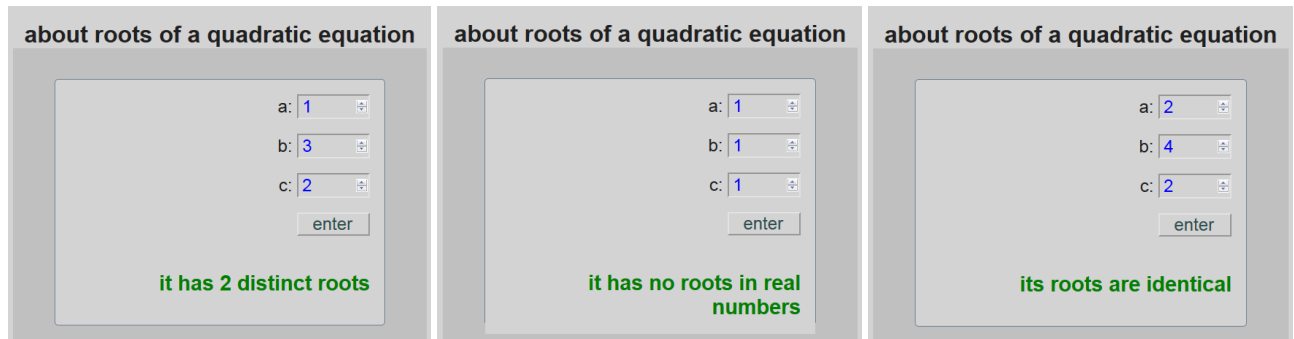
- Make sure name of the function is `equation()`
- Replace the lines you had for heron’s formula; with what you have in your **img\_03.jpg**
- Correct the output to have a more relevant label and correct value as the picture above shows

Again, note that the answer is fixed to 2 digits after the decimal point. Also note that roots are shown in different lines.

Before moving on to Exercise 4, compare your code in **ct\_Ex3.js** with your **img\_03.jpg**, your flowchart for this problem. Make sure they match.

**Exercise 4.** Copy **ct\_Ex3.html** and **ct\_Ex3.js** to new files named **ct\_Ex4.html** and **ct\_Ex4.js**.

In this exercise, you should translate your flowchart of Exercise 4 of Part 1 to js. You should make some changes in both **html** and **js** files, such that you get one of the following results depending on the inputs.



The image shows three side-by-side screenshots of a web form titled "about roots of a quadratic equation". Each screenshot shows the same form with input fields for 'a', 'b', and 'c', and an 'enter' button. The results displayed below the inputs are:

- Left screenshot:** a: 1, b: 3, c: 2. Result: "it has 2 distinct roots".
- Middle screenshot:** a: 1, b: 1, c: 1. Result: "it has no roots in real numbers".
- Right screenshot:** a: 2, b: 4, c: 2. Result: "its roots are identical".

In particular, you need to make 2 changes in your **ct\_Ex4.html** as follows:

- Make sure you link your **html** file to your new **js** file
- Change the header to be "about roots of a quadratic equation"

Also, you need to make 2 changes in your **ct\_Ex4.js** as follows:

- Replace the lines you had for calculating the roots with what you have in your **img\_04.jpg** to determine if the equation has one, two, or no roots in real numbers
- Correct the output to reflect the message

Before moving on to Exercise 5, compare your code in **ct\_Ex4.js** with your **img\_04.jpg**, your flowchart for this problem. Make sure they match.

**Exercise 5.** Copy **ct\_Ex4.html** and **ct\_Ex4.js** to new files named **ct\_Ex5.html** and **ct\_Ex5.js**.

In this exercise, you should translate your flowchart of Exercise 5 of Part 1 to js.

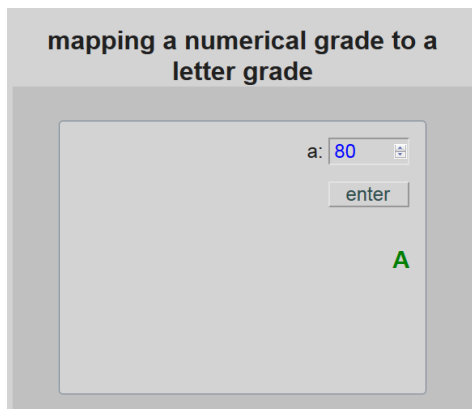
In particular, you need to make 3 changes in your **ct\_Ex5.html** as follows:

- Make sure you link your html file to your new js file
- Change the header to be “mapping a numerical grade to a letter grade”
- When the “enter” button is clicked, the event handler `mapping()` should be triggered.

Also, you need to make 2 changes in your **ct\_Ex5.js** as follows:

- Make sure name of the function is `mapping()`
- Remove codes from line 9 to the comment above the `switch` statement
- Uncomment the block that the `switch` statement is in

Now, save the changes and launch **ct\_Ex5.html** with your browser. If you enter any number greater than 79, it is correctly mapped to an A or A+; Also, if you enter any number less than 48, it is correctly mapped to F. But, if you enter, any number between 40 and 79, it’s not mapped correctly. Your task is to fix this issue.



In particular, you need to make one big change in your **ct\_Ex5.js** as follows: add seven more cases to the `switch` statement to map other grades to B+, B, C+, C, D+, D, and E correctly.

Now, compare your code in **ct\_Ex5.js** with your **img\_05.jpg**, your flowchart for this problem. Make sure they match. That means if in your flowchart, you used several `if` statements instead of a `switch`, you should change either your js code or your flowchart.

### Exercise 6 (further practice, for up to 10 % bonus):

You may want to continue this project and add the other flowcharts (from exercises 6–8 of Part 1) and the corresponding js implementations.

**Hint:** Even though in the flowcharts of exercise 6–8, you probably have `while` loops, you do not need to use the `while` loop in js because your code is *event-driven*: every time the button is clicked, a new iteration of the loop commences.

### F. AFTER-LAB TASKS (THIS PART WILL NOT BE GRADED)

In order to review what you have learned in this lab, as well as for expanding your skills further, we recommend the following questions and extra exercises:

- 1) You should revisit the 8 exercises after the lab and learn about the parts of your flowcharts that were not a good match to your js code.
  - a. You may want to do some sort of reverse engineering here: study the js code that you eventually developed for each exercise and draw a flowchart for that. Be careful not to include any JavaScript specific notation in your flowcharts. Flowcharts should be as independent as possible from any particular programming language. That means your flowchart should be understandable to anyone who knows programming even if he/she does not know any JavaScript.
  - b. Hence, your flowchart should not use functions like `parseFloat`, `getElementById`, etc., or keywords like `if`, `else`, `var`, etc. You are also highly discouraged to use “=” as an assignment operator; use “←” instead.
- 2) Once you have your own 8 flowcharts and the corresponding js implementations polished, take photos (or screenshots) of each and add them to your myLearningKit webpage such that when buttons 1 to 8 are clicked, your corresponding solutions are shown.
- 3) We will provide you with sample solutions on Oct 11. The sample solution should NOT be used as a means of learning how to tackle those 8 exercises. Instead, it should be only used as a reference to compare your own solution with our solution and learn from differences. In general, you cannot learn much computational thinking skills, if any, by studying a solution without putting your efforts first to come up with one yourself (even if your solution is not perfect). As an analogy, no one can learn how to ride a bicycle practically by watching (even 100s of hours of) how others do it. This is true for many other skills, including computational thinking.

Please feel free to discuss any of these questions in the course forum or see the TAs and/or Instructors for help.