

EECS 1012: LAB 04 –HTML+JS+ComputationalThinking

A. REMINDERS

- 1) You should 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 during your lab hours.
- 2) You are required to pass the pre-lab mini quiz posted on eClass not later than the first 10 minutes of your lab time. **You should study the recent course materials and corresponding links/hints and Section B of this document, as well as working on at least first tasks of this lab before trying the prelab quiz.** You have 4 attempts; and you need at least 80% to pass. However, each time you may get some different questions. You should try your first attempt at least one day before your deadline so that, if needed, you have time to (re)study the materials for your next attempts. Failing the pre-lab mini quiz is equal to failing the whole lab, yet you are still highly encouraged to complete the lab and submit your work to eClass.
- 3) 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.
- 4) You can submit your lab work in eClass any time before 21:00 on Wednesday of the week the lab is for. In order to pass this lab, your grade in it should be at least 70%.

B. IMPORTANT PRE-LAB WORKS YOU NEED TO DO BEFORE GOING TO THE LAB

- 1) Download this lab files and read them carefully to the end.
- 2) You should have a good understanding of
 - Events (such as `onclick`) and event handlers
 - `document.getElementById().innerHTML`
 - js functions such as `parseInt()`, `parseFloat()`, `toFixed()`
 - the `if` statement and `switch` statement in js
 - memory space (aka variable), js operators such as `+` and the concept of overloading
 - flowchart symbols as described in the lecture notes
- 3) Practice drawing flowchart symbols in `draw.io` or 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 computation problems, followed by implementation in JS

D. TASKS

- 1) Your first and major task in this lab is to design eight algorithms and draw the corresponding flowcharts. This task must be done in teams of two. (By the permission of your Lab TA, only one team can have three members if the lab population is odd.) While you are done, you should show your flowcharts to the TAs before you go to next part. The TA may ask you to make minor modifications to your flowcharts to demonstrate your computational thinking skills in those contexts.
 - Note you should NOT open VS-Code or browsers before finishing Task 1. You can draw your flowcharts on paper or draw them in `draw.io` or in MS Word or PowerPoint.
- 2) Then, you are provided with `ct.html` document and supporting files such as `ct.css` and `ct.js`. Your task is to translate your first 5 flowcharts to js.
- 3) You will generate at least five `html` and `js` files in this process. You should demo each HTML file to the TA. For that, please, have each `html` file open in a different tab so you can show the progression.
- 4) See next pages for details on how to modify your `html` and `js` files.

E. SUBMISSIONS

- 1) Manual verification by a TA

You may want to have your TA verifying 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 (or tar.gz) and the compressed file submitted.

Make sure size of each image file is not more than 1MB, otherwise you may not be able to upload your work.

F. COMPUTATIONAL THINKING

Part 1: This part must be done in teams of two. (By the permission of the TA, only one team can have three members if the lab population is odd.) If you have done it at home, you are required to discuss it with a peer from your lab before you show your final solution to your TA.

Using a computer program (or on paper), draw the following flowcharts and write your name on each. By end of this lab, you should take a screenshot (or a picture) from each flowchart and both you and your teammate should submit them to eClass as `img_{01,02,03,04,05,06,07,08}.jpg` files, where `img_x` is the flowchart for exercise `x` below. Make sure the size of each image is less than 500KB, e.g. by reducing the resolution of your camera.

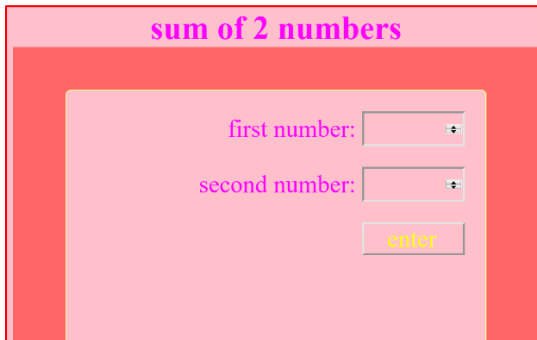
IMPORTANT: You are required to use the symbols introduced in the lecture which are inspired from this book (“Computer Science: a first course” by Forsythe, Keenan, Organick, Stenberg)

IMPORTANT: You are required to provide preconditions and postconditions for each solution you provide.

- Ex 1) draw a flowchart for a computer program to receive two numbers and output their sum.
- 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 sides of a triangle and, by using Heron’s formula, calculate and output the area of the triangle. You may need to refresh your memory or learn about Heron’s formula by visiting <https://www.mathopenref.com/heronsformula.html>
- 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. Write a precondition that assumes coefficients are good enough such that a solution in real number exists. That means your design should not concern cases for which 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 determines 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 40 to 49, it’s mapped to 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 program should output how many positive and how many negative numbers have been entered, then it stops.
- 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 modulus operator (%) over 6 to verify the remainder whether the number is divided by 6. You can do any other math trick you wish.

Part 2: you are given **ct.html**, **ct.css**, **ct.js** files. Reading these files carefully in order to enhance your learning before doing the following exercises.



The screenshot shows a web form with a pink header bar containing the text "sum of 2 numbers". Below the header is a light pink rectangular area. Inside this area, there are two input fields. The first is labeled "first number:" and the second is labeled "second number:". Both fields are empty. Below the second input field is a button labeled "enter".

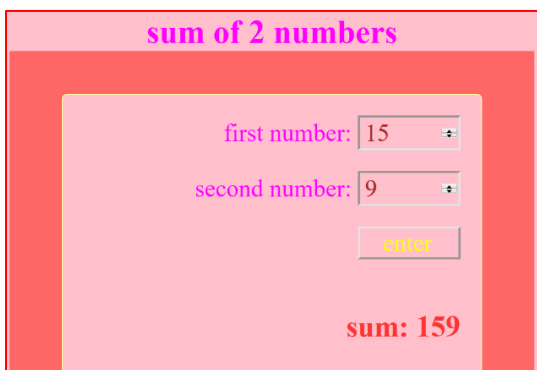
Exercise 1. Copy **ct.html** to a new file named **ct_Ex1.html**. Copy **ct.js** to a new file named **ct_Ex1.js**.

Launch **ct_Ex1.html** with your browser and enter two numbers and click on the “enter” button, nothing happens. In this exercise, you would modify the code such that it runs properly:

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 `add()` 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 15 and 9. You should see the following result:



The screenshot shows the same web form as before, but now the input fields contain the numbers "15" and "9". Below the "enter" button, the text "sum: 159" is displayed in red.

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 “string” too.

Also, the “+” operator is overloaded in `js` (and in 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 “15” and “9”, therefore “+” concatenates them, and the result is “159”. `parseInt()` is a function that receives a string as its argument and returns its equivalent integer number. In other words, it can receive “15” and return 15. It can also receive “9” and return 9.

In summary, “15”+“9” results in “159”. But, 15+9 results in 24. So, open your **ct_Ex1.js** file and by using `parseInt`, modify the code such that sum of the two inputs are calculated, not their concatenations.

sum of 2 numbers

first number: 15

second number: 9

enter

sum: 24

Before going to Exercise 2, compare the code in **ct_Ex1.js** with your **img_01.jpg**, i.e., the flowchart you drew in Exercise 1 of Part 1. Are they conceptually the same? If the answer is “no”, something may be 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 normally 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 involve in how w and h should be inputted. The designer just states to input w and h. Its the task of the programmer to translate the flowchart to an appropriate statement in the target programming language, here js; elsewhere, java, python, c, etc. This is true for many other components 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 sides of the triangle.

area of a triangle when you enter three good sides

a: 10

b: 11

c: 4

enter

area: 19.96

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 “area of a triangle when you enter three good sides”
- 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 side of a triangle cannot be negative numbers change the min and max to 1 and 100

f) When the “enter” button is clicked, the event handler `area()` should be triggered.

Also, you need to make about 5 changes in your **ct_Ex2.js** as follows:

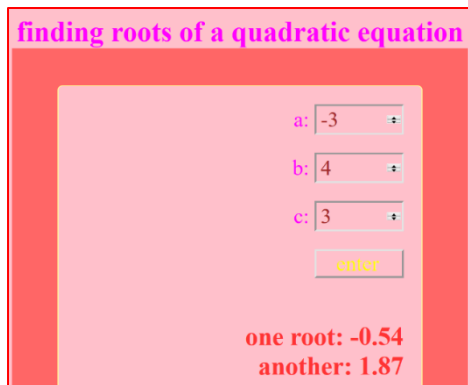
- Make sure name of the function is `area()`
- Make sure pre/post conditions are updated to reflect how this function works
- Add a line to capture the value of `c`
- Replace the line `s=w+h;` with what you have in your flowchart for Heron’s formula
- Correct the output to have a more relevant label and correct value as the picture above illustrates

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

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 are 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.



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 “finding 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 4 changes in your **ct_Ex3.js** as follows:

- Make sure name of the function is `equation()`
- Make sure pre/post conditions are updated to reflect how this function works
- 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 are 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.

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 3 changes in your **ct_Ex4.js** as follows:

- Make sure pre/post conditions are updated to reflect how this function works
- 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 are 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 4 changes in your **ct_Ex5.js** as follows:

- Make sure name of the function is `mapping()`
- Make sure pre/post conditions are updated to reflect how this function works
- 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 40, 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 are 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):

Only If you have done previous tasks perfectly, you may want to continue this project and add at least two more flowcharts (from exercises 6-8 of part 1) and corresponding **js** implementations.

Hint. Even though in the flowcharts of exercises 6-8, you may have **while** loops, you do not really need them in **js** because your code is event-driven: every time the button is clicked, next iteration of the loop is conducted. Loops will be covered in next lectures.

G. AFTER-LAB WORKS (THIS PART WILL NOT BE GRADED)

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

- 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 programming languages. 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 `"="` for an assignment; instead you should use `"←"`.
- 2) Once you have your own 8 flowcharts and corresponding **js** codes 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 Feb 5. 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 a solution (even a non-perfect one). 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.