

EECS 1012: LAB 5 – More Computational Thinking (June 10–17, 2021)

A. REMINDERS

- The pre-lab mini quiz is considered part of this lab.
- You are required to complete the pre-lab mini quiz 4 posted on eClass no later than the beginning of your lab time, i.e., 10am this Friday if you are in the morning sessions (lab03, 04), and 1:30pm if you are in the afternoon sessions (lab01, 02, 05)
- You are welcome to attend the lab session, if you stuck on any of the steps below. TAs and instructor will be available via Zoom to help you. If you need help, you are expected to come to your own lab session. Please check the updated lab times and Zoom links on eClass (may change slightly for different labs).
- Please be patient while you are waiting to be admitted by the zoom host. This is one-to-one session and the TAs admit on first come first served basis.
- Instructor will be around in different lab sessions to help. If you need to talk to the instructor, let the TA know and they would notify the instructor.
- You can submit your lab work anytime before the specified deadline.

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

- 1) Download this lab's files and read them carefully to the end.
- 2) See the sample solution to Lab04.
- 3) You should have a good understanding of
 - Iterations: *while*, *do-while* and *for* loop symbols as introduced in the lecture notes
 - JS operator for division and modulus operator
 - JS *prompt* box; as well as *console.log* for debugging purposes when needed
- 4) Practice drawing flowchart symbols in `draw.io` or MS PowerPoint or Word. If you plan to draw your flowcharts on paper, please make sure you have pencils, erasers, and perhaps a ruler to make it neat.

C. GOALS/OUTCOMES FOR LAB

- To practice more on computational thinking and implement the solutions in JavaScript. The focus of this lab is on loops.

D. TASKS

- 1) You first and major task in this lab is to draw seven flowcharts.
 - 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 PowerPoint or Word.
 - You are encouraged to discuss the flowcharts with peers, TAs and instructor. **But you cannot copy other's solutions.**
- 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 seven flowcharts to **JavaScript** code.
 - You will generate seven **html** and **js** files in this process.
- 3) See the following few pages for details on how to modify your **html** and **js** files.

E. SUBMISSION

eClass submission. More information can be found at the end of this document.

F. COMPUTATIONAL THINKING

Part 1: For this flowchart part you are encouraged to discuss with other people (peers, TAs, instructor), even to share your partial solutions. But you should not copy other people's solution.

Using a computer program (or a pen and pencil), draw the following flowcharts and write your name on each. By the end of this lab, you should take a screenshot (or a picture) from each flowchart and you should submit them to EClass as `img_{07,08,09,10,11A,11B,12, 13}.jpg` files, where `img_x` is the flowchart of exercise `x` below. Try to keep the size of each image below 500 KB, e.g., by reducing the resolution of your camera.

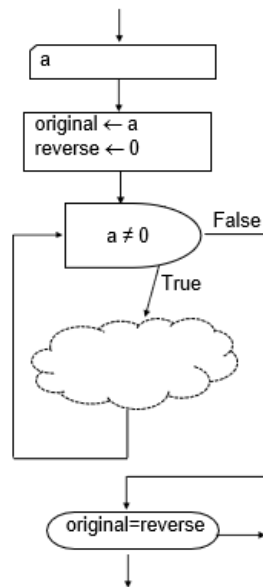
IMPORTANT: You are required to use the symbols introduced in the lecture which are inspired from this book ("Computer Science: a first course" by Forsythe, Keenan, Organick, Stenberg).

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

IMPORTANT: In Ex8 to 10, you are not allowed to use strings to process the input number. Instead, you should work with numbers and math operators, such as division, modulus, etc.

- Ex 7) Devise a flowchart to keep on receiving an integer number from user and output whether the number is positive or negative. The process repeats until a 0 is entered, then output "program ended" and ends.
- Ex 8) Devise a flowchart to receive a positive integer number and output each digit separately.
For instance, if the input is 692, the program should output 6, 9, 2. Another example, if the number is 135429 the program should output 1, 3, 5, 4, 2 and 9.
Hint: as demonstrated in class, the key idea is that in the loop of your flowchart, separate the digits along the way—i.e., in loop iterations. In each iteration, you can output the digit or store the digit into a string which will be outputted after the loop.
For instance, assume the input number, stored in a , is 235, then in the first iteration of the loop, you get digit 5 from the input number a (how?) and output "5" or store "5" in a string. a is reduced from 235 to 23 (how?). Now, in the second iteration of the loop, you get digit 3 from 23, and a is reduced from 23 to 2. You update output to be "3, 5" or update the string to be "3, 5". You repeat the steps until a is reduced to 0 or less. Then if you have used string to store result, you output the string.
- Ex 9) Devise a flowchart to receive a positive integer number and output how many of its digits are equal to 3 or 7.
For instance, if the input is 772, the program should output 2, because there are two 7 there. In another example, if the input is 14367, the program should output 2 as there is a 3 and a 7 in it.
Hint, use logical OR in the condition, that is $a = 7 \text{ OR } a = 3$.
Hint, before entering the loop, declare a counter variable and assign 0 to it. Inside the loop, after you separated a digit check if it's equal to 7 or 3, or not. If yes increment the counter variable. Output the value of the counter variable after the loop.
- Ex 10) Devise a flowchart to receive a positive integer number and output "yes" if it's equal to its reverse; otherwise, output "no". For instance, if the input is 63936, the program should output "yes", because if you read the digits from left to right or from right to left, it's the same number. But, if the input is 632, the program should output "no" because 632 is not the same as 236.
Hint: the idea is that you create the reverse version of the original number, and finally compare the original and reserve version of the numbers. Specifically, in the loop of your flowchart, you may separate the digits and make the reverse of the input number along the way—i.e., in loop iterations. For instance, assume the input

number, stored in a , is 235, before entering the loop, you could declare a variable *reverse* and initialize it to 0 as well as storing the input number 235 in a variable, let's call it *original*. Then in the first iteration of the loop, you get digit 5 from the input number a (how?), and you store value 5 to *reverse*, and a is reduced from 235 to 23. Now, in the second iteration of the loop, you get digit 3 from 23, and somehow the *reverse* number should be updated from 5 to 53, (hint: $5 \cdot 10 + 3$) and input a is reduced from 23 to 2. Then, in the third iteration of the loop, you get digit 2 from 2, and *reverse* is updated from 53 to 532 now and a is reduced from 2 to 0 or below, then you stop the loop. After the loop, you compare your *reversed* value with the value stored in *original* and see if they are the same. (Alternatively, instead of a , you can process on *original*, and finally compare *reverse* with a with.) Below, we show you part of a flowchart that you may have for this exercise. You should be able to write the body of the loop based on example that we explained above.



Ex 11A) Devise an algorithm to receive a positive integer number, as n , and output $n!$ (n Factorial). At the end the algorithm output " $x! = y$ " where x is the input value and y is the factorial value. For example, for input 6, output " $6! = 720$ ".

The major difference between the flowchart of this exercise compared to those of previous exercises is that the number of iterations in this exercise is *deterministic*. In other words, we know how many iterations needs to be made—in advance. For instance, for calculating $5!$, five iterations is required. No more no less, whereas in previous exercises we did not know how many digits the input number will have. Therefore, we did not know how many iterations the loop will need. Hence, none of the previous exercises should have been solved with a **for** loop. This exercise, though, should be devised with a **for** loop.

Ex 11B) Devise an algorithm to receive a positive integer number, as n , and output $n!$ (n Factorial). Use **while** loop. Although **for** loop, as you did in part A, fit this question well, here we use **while** loop to implement (every **for** loop can be rewritten to a **while** loop)

Ex 12) Devise an algorithm to input an integer greater than 1, as n , and output the first n values of the [Fibonacci sequence](#). In Fibonacci sequence, the first two values are 0 and 1 and other values are sum of the two values preceding it. For instance, if the input is 4, the program should print 0, 1, 1, 2,. As another example, if the input is 9, the program should output 0, 1, 1, 2, 3, 5, 8, 13, 21,.

This exercise can be done with a **for** loop too, because—as an example—if the input is 10, the loop should iterate exactly 8 times. Note that for the first two values, we do not need to iterate; we already know they

are 0 and 1.

After you are done with this exercise, learn more about applications of Fibonacci number in real life here: [http://www.ijesi.org/papers/Vol\(6\)9/Version-3/B0609030714.pdf](http://www.ijesi.org/papers/Vol(6)9/Version-3/B0609030714.pdf) . There might be a question on this in the pre-lab mini quiz.

Hint: As a new value, in Fibonacci sequence, can be constructed by sum of the last and 2nd last values, you may want to use three variables:

```
secondLast=0;
last=1;
newValue;
```

Note that secondLast and last variables are initially 0 and 1, respectively, because that's they way Fibonacci sequence starts.

In the body of the for loop, we calculate newValue (how?) and update the values for last and secondLast variables – last is updated with newValue, and secondLast is updated with last value.

- Ex13) Devise an algorithm to input a positive integer, n , – and by using [and] characters – output the figure below that has n rows and each row k has k pairs of []. For instance, if input is 5, the figure on the left should be generated by the program. If the input is 8, the figure on the right should be generated.
- Hint: you need to use nested loop, i.e., a loop that contains another (inner) loop. A similar problem was demonstrated in class.

number: 5	number: 8
	[]
	[] []
[]	[] [] []
[] []	[] [] [] []
[] [] []	[] [] [] [] []
[] [] [] []	[] [] [] [] [] []
[] [] [] [] []	[] [] [] [] [] [] []

Part 2: you are given **ct.html**, **ct.css**, **ct.js** files. By reading these files carefully, you can enhance your learning before diving to details of exercises below. In this part, you translate your flowcharts of Part 1 to JavaScript code.

Exercise 7. Copy **ct.html** to a new file named **ct_Ex7.html**. Copy **ct.js** to a new file named **ct_Ex7.js**.

Launch **ct_Ex7.html** with your browser and click on the button, nothing happens. In this exercise, you translate your flowchart of Ex7 of Part 1 to its equivalent JavaScript code. First let's fix some html code:

Make changes to **ct_Ex7.html**, including (but may not limit to)

- Connect it to **ct_Ex7.js** by fixing the script tag in the head element.
- Chang the header of the document to show "receiving integers until 0"
- Correct the name of the event function of the button to be "*readInteger()*"
- Add your name to the list of authors of this page.

Also, in your **ct_Ex7.js**, implement function *readInteger()*. You should use your flowchart that you drew in Part 1 to

complete the function. In your flowchart, you should have a loop, perhaps a while loop (or a do-while).



Exercise 8. Copy **ct_Ex7.html** to a new file named **ct_Ex8.html**. Copy **ct_Ex7.js** to a new file named **ct_Ex8.js**.

In this exercise, you translate your flowchart of Ex 8 of Part 1 to its equivalent JavaScript code. First let's fix some html code. Make four changes to **ct_Ex8.html**, as follows:

- Connect it to **ct_Ex8.js** by fixing the script tag in the head element.
- Correct the header texts of the document to show "Separating Digits of a Positive Integer"
- Correct the name of the event function of the button to be "*separateDigits()*"

Also, in your **ct_Ex8.js**,

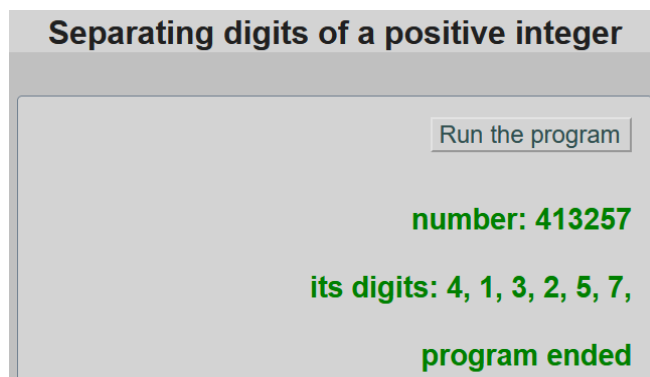
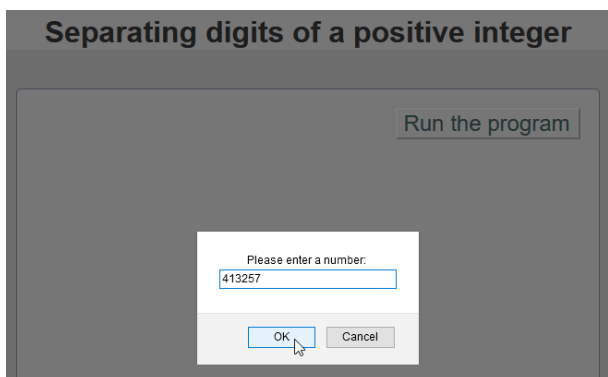
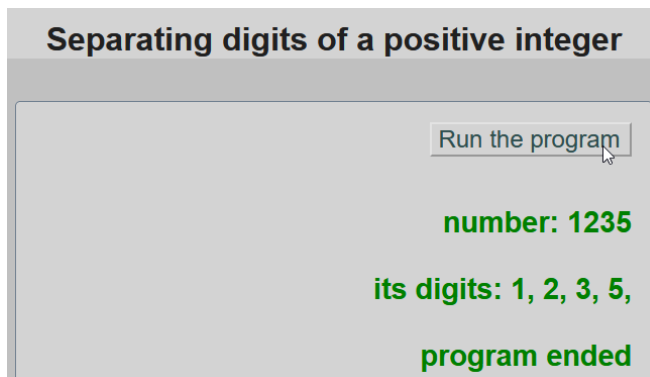
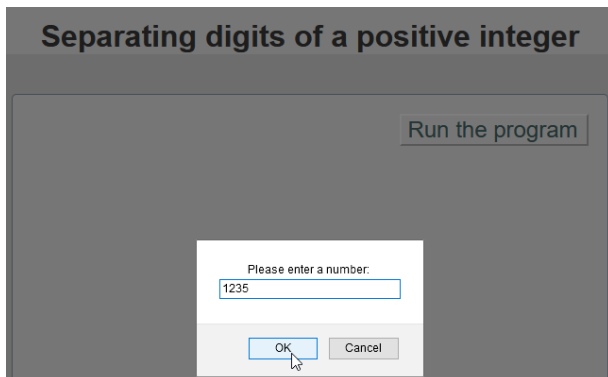
- Name of the function should be *separateDigits()* ()
- Uncomment the line that sets the output message to "*number: " + a + "

its digits:"*"
- Implement the function based on your flowchart that you drew in Part 1

Hint: in JS when you divide two integers, or two strings of numbers, such as 27/10, you get 2.7 You may need the result to be 2 (but not round to 3). A function that you used in the previous labs can be helpful. You can also search how to convert floating point number to integer in JS for more solutions.

Hint: if you stored the digit in a string, then when you get a new digit, append the new digit in front of the string. Then output the string after the loop. Alternatively, if you output the result in each iteration, then you update the existing texts by retrieving the exiting texts, and append the new digit in front.

Once you are done, run the program, you should see the following results if you enter 1235 or 413257. Test your program with more inputs. If the program does not give the expected results, debug your code by inspecting the page and examine the console output, as shown in class.



Before going to next exercise, make sure your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 9. Copy **ct_Ex8.html** and **ct_Ex8.js** to new files named **ct_Ex9.html** and **ct_Ex9.js**.

In this exercise, you translate your flowchart of Ex9 of Part 1 to its equivalent JavaScript code.

You need to make changes in your **ct_EX9.html** including: js script link, header text, and name of the event function *digits37()*.

In the js file of previous exercise, you (should have) separated digits of a number by modulus and division operators. Now, in **ct_Ex9.js**, you need to make the following changes

- Name of the function should be *digits37()*
- Change the output message from “its digits” to “number of 3 or 7’s”
- Implement the function based on your flowchart.

Once you are done, run the program, you should see the following results if you enter 789041 or 5323653345. If not, debug your code.

# of 3 or 7's in digits of a positive number	# of 3 or 7's in digits of a positive number
<div>Run the program</div> <p>number: 7425</p> <p>number of 3 or 7's: 1</p> <p>program ended</p>	<div>Run the program</div> <p>number: 327507472</p> <p>number of 3 or 7's: 4</p> <p>program ended</p>

Before going to next exercise, make sure your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 10. Copy **ct_Ex9.html** and **ct_Ex10.js** to new files named **ct_Ex9.html** and **ct_Ex10.js**.

In this exercise, you translate your flowchart of Ex 10 of Part 1 to its equivalent JavaScript code.

The event function is *reverseSame()*.

Several changes that you need to make here are minor changes like what you did in Exercise 10 in **ct_Ex10.html** and **ct_Ex10.js**. So, we do not re-state them again here, to encourage you to make such changes independently and with minimum guidance.

Once you are done, run the program, you should see the following result if you enter 23432 or 897. Test your program with more inputs. If the program does not give the expected results. Debug your code by inspecting the page.

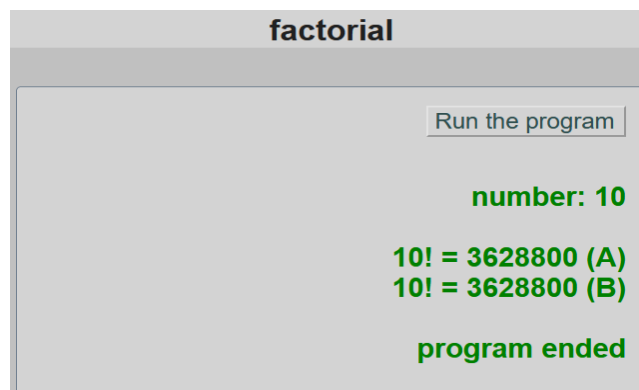
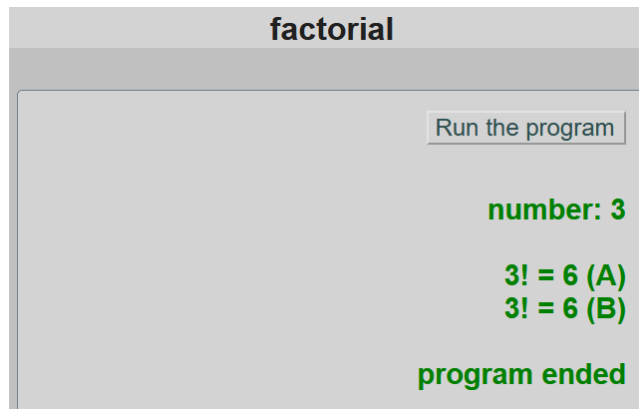
a number equal to its reverse	a number equal to its reverse
<div>run the program</div> <p>number: 23432</p> <p>equal to its reverse? yes</p> <p>program ended</p>	<div>run the program</div> <p>number: 897</p> <p>equal to its reverse? no</p> <p>program ended</p>

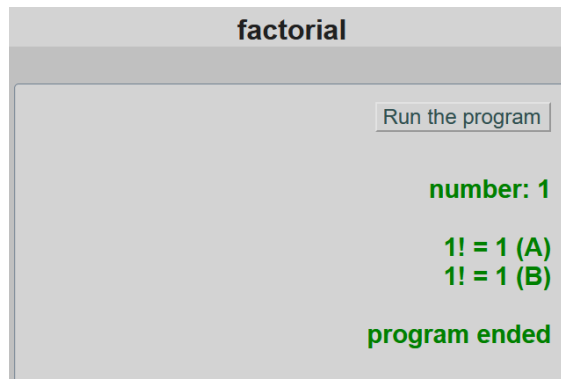
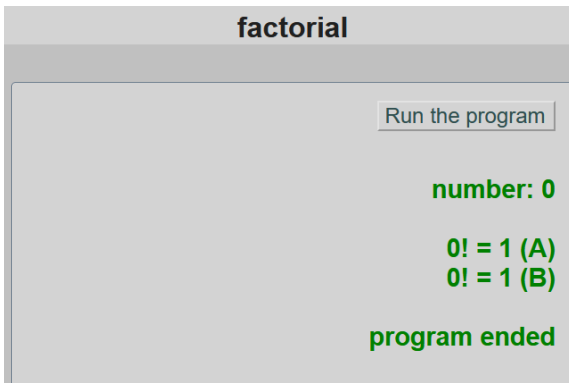
Before going to next exercise, make sure your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 11. Copy `ct_Ex10.html` and `ct_Ex10.js` to new files named `ct_Ex11.html` and `ct_Ex11.js`. In this exercise, you translate your flowchart of Ex 11A and Ex11B of Part 1 to its equivalent JavaScript code. This exercise also gives you a heads-up on calling sub-program, which we will cover in class soon. In **html**, change the event function to `factorial_A()`. Make other necessary changes such as changing header texts. In **JS** file, make the following changes.

- Define function `factorial_A()`. This function reads user input from prompt, and calculates the factorial using **for** loop. This function implements the flowchart you did in Ex13A.
 - Store the input in a variable `num`.
 - Calculate using `for` loop.
 - After calculating, output the factorial, ended by "(A)".
 - after this, add line `factorial_B(a)` ; This calls another function `factorial_B()` with parameter `a`.
 - After this function call, outputs "program ended" (update the existing texts that were just updated in `factorial_B()`).
- Outside of `factorial_A()`, define another function `factorial_B(a)`, which implements the flowchart you did in Ex13B.
 - In `factorial_B(a)`, calculate factorial of `num` using `while` loop. **Note that this function does not read user input from prompt**, as it already got input from its parameter `num`. After calculation, output the result, ended by "(B)" (update the existing texts produced in `factorial_A()`).

Once you are done, run the program, you should see the following result. Test your program with more inputs. For any input number, both the two functions should produce the same factorial value. If not, debug your code.





Before going to next exercise, make sure your JS code is a good match to your flowcharts for this problem. If not, you should fix this mismatch.

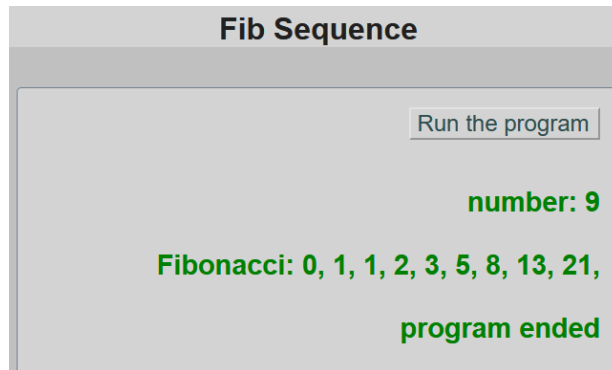
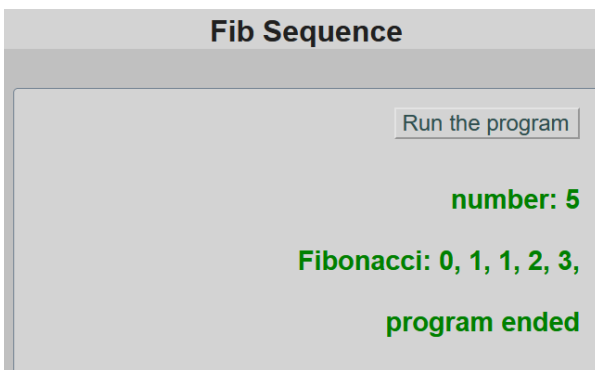
Exercise 12. Copy `ct_Ex11.html` and `ct_Ex11.js` to new files named `ct_Ex12.html` and `ct_Ex12.js`.

In this exercise, you translate your flowchart of Ex12 of Part 1 to its equivalent JavaScript code.

Several changes that you need to make here are minor changes like what you did in previous exercises in `html` and `js` files. So, we do not re-state them again here. Just make sure name of the event function is `fibonacci()`, both in your `html` and `js` files.

Once you are done, run the program, you should see the following result if you enter 5 or 9. Test your program with more inputs. Debug your code if the program does not give the expected results.

Before going to next exercise, make sure your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.



Exercise 13. Copy `ct_Ex12.html` and `ct_Ex12.js` to new files named `ct_Ex13.html` and `ct_Ex13.js`.

In this exercise, you translate your flowchart of Ex 13 of Part 1 to its equivalent JavaScript code.

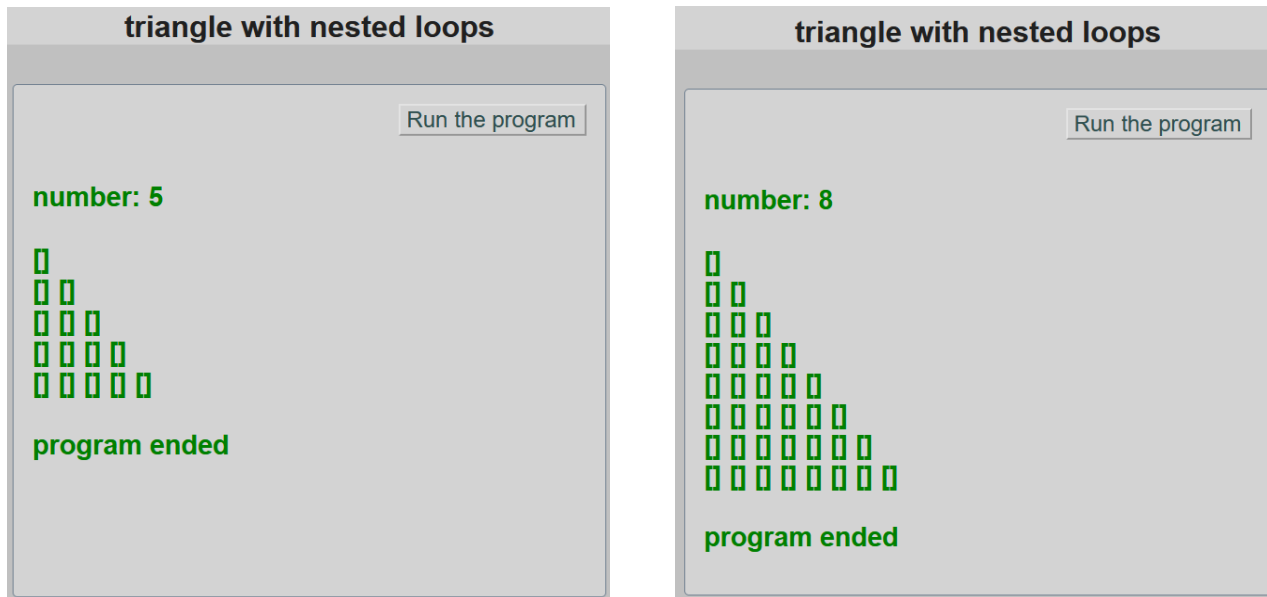
Several changes that you need to make here are minor changes like what you did in previous exercises in `html` and `js` files. So, we do not re-state them here.

This exercise can be done with `for` loop, as iterations are deterministic. For instance, if input is 5, we need to iterate exactly 5 times to output those 5 rows; also, in each row i , we should put i pairs of `[]`. So, we need to nest one loop inside another.

Note: in this exercise you would need to change text alignment to left. You can do this just in your `js` code by modifying the `style` property of `outputObj` just before you enter the loop.

Once you are done, run the program, you should see the following result. Test your program with more inputs. Debug

your code if the program does not give the expected results.



Note: if you have difficulties understanding and implementing nest loops, you can play with the lecture program posted on eClass, which uses nested loops to print square.

G. AFTER-LAB TASKS (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 practice:

- 1) You should revisit the 7 exercises after the lab and learn about the parts of your flowcharts that were not a good match to your JavaScript code.
 - a. You may want to do some sort of reverse engineering here: study the JavaScript 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 JavaScript code polished, take photos (or screenshots) of each and add them to your myLearningKit webpage such that when the corresponding buttons (Problem07~Problem13) are clicked, your solutions are shown. For Problem11 (factorial), use one of the flowchart for 'design' image and put one of the two JS solutions to the 'JavaScript solution' image, and another JS solution to the 'Another solution' image.
- 3) 'JavaScript solution' image, and another JS solution to the 'Another solution' image.
- 4) We will provide you with sample solutions soon. The sample solution should NOT be used as a means of learning how to tackle those 7 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). In other words, **you learn Computational Thinking by actually doing it** not by reading it or watching examples. Last but not the least, students who do not practice Computational Thinking **independently or adequately** will feel a lot of time pressure during all upcoming tests and the final exam that would result in

low grades.

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

H. Submissions.

You should already have a **Lab05** folder that contains the following files:

ct.html, ct.js, ct.css

ct_Ex{7,8,9,10,11,12,13}.html and **ct_Ex{7,8,9,10,11,12,13}.js**

If you haven't done so, copy the 8 images of your flowcharts to this folder as well

img_{07,08,09,10,11A,11B,12,13}.jpg

Make sure size of each image file is not more than 0.5MB, otherwise you may not be able to upload your work

Your HTML should pass the HTML validator at <https://validator.w3.org>

Compress the Lab05 folder (zip or tar.gz), and then submit the (single) compressed file on eClass.