EECS 1710 - F2022
LAB 4 :: Loops & Arrays

Prerequisite (labs 0-3) – please ensure that you have setup your EECS account, and can log into the lab machines prior to starting this lab, and you are familiar with submit/websubmit.

NOTE: This lab is worth 3% of your final grade.

STEP 1: Importing and unzipping the starter code

Download and extract the following lab4 starter code:
http://www.eecs.yorku.ca/course_archive/2022-23/F/1710/labs/lab4/lab4.zip

There are three files to submit for this lab. Assuming the above zip file is downloaded and extracted into the `1710/labs/` folder in your home directory, you should be able to navigate to the labs folder and submit everything with the following commands:

```
cd
cd 1710/labs/
submit 1710 lab4 lab4/*
```

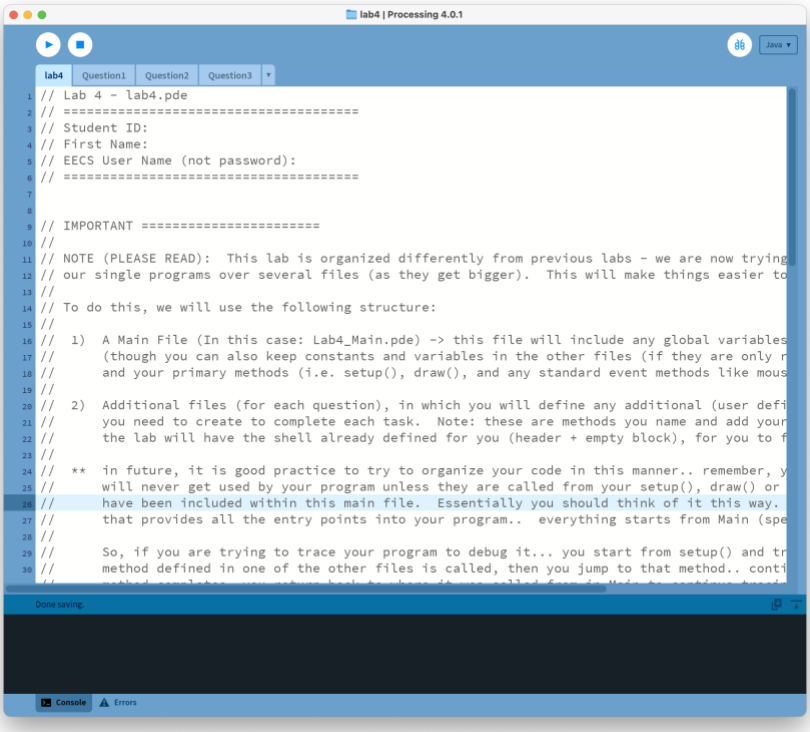
This will submit all files within the lab4 folder. Alternatively, you may use web submit to submit/delete individual files (see link at the end of this document). To list files submitted, type:

```
submit -l 1710 lab4
```

STEP 2: Exercises

Please note, **the organization of your starter code is different in this lab**. As our programs grow in size, they can benefit from being split over multiple files. In the past, you have worked on individual processing sketches that essentially have one file. From now on we will begin to look at processing sketches that include several files.

The sketch itself has a name associated with it, which also happens to be the name of the folder, and one main file. In this lab, the sketch is called "**`lab4`**", and this is the same name as the folder within, when you extract **`lab4.zip`**. Other files can be included in the same sketch folder (they can be named anything, and are added when you hit the dropdown menu next to the *sketch name -> New Tab*:



Note that the other files will show up as tabs (`Question1, Question2, ...`), when you open the **`lab4`** sketch. For all of these to work, we cannot treat them as independent files. This means that **ONLY ONE** of the files can have the primary methods defined within (i.e. `setup()`, `draw()`, and any other method files for that matter).. i.e. we can't define a `setup()` method in each file, because then there would be two conflicting methods with the same name & signature in your program!

From now on, we will use the ***convention*** that we will try to keep all the main methods (`setup()`, `draw()` and any event methods like `mousePressed()` etc., that processing already recognizes as standard – entirely within our main file for the program – the file that has the same name as the sketch (in this lab, it is **`lab4.pde`**). In the other files, we will define any other methods that our program will use.

-> Please **read the comments** in the `lab4.pde` before continuing in this lab.

Question01 (Straight-Line Curves / String Art)

In **`Question1.pde`**, you will write several methods that will iterate (i.e. use a for or while loop – see lectures from week 5), over a range of start and finish points (along one or more edges of the app window), to create curved patterns from straight lines:

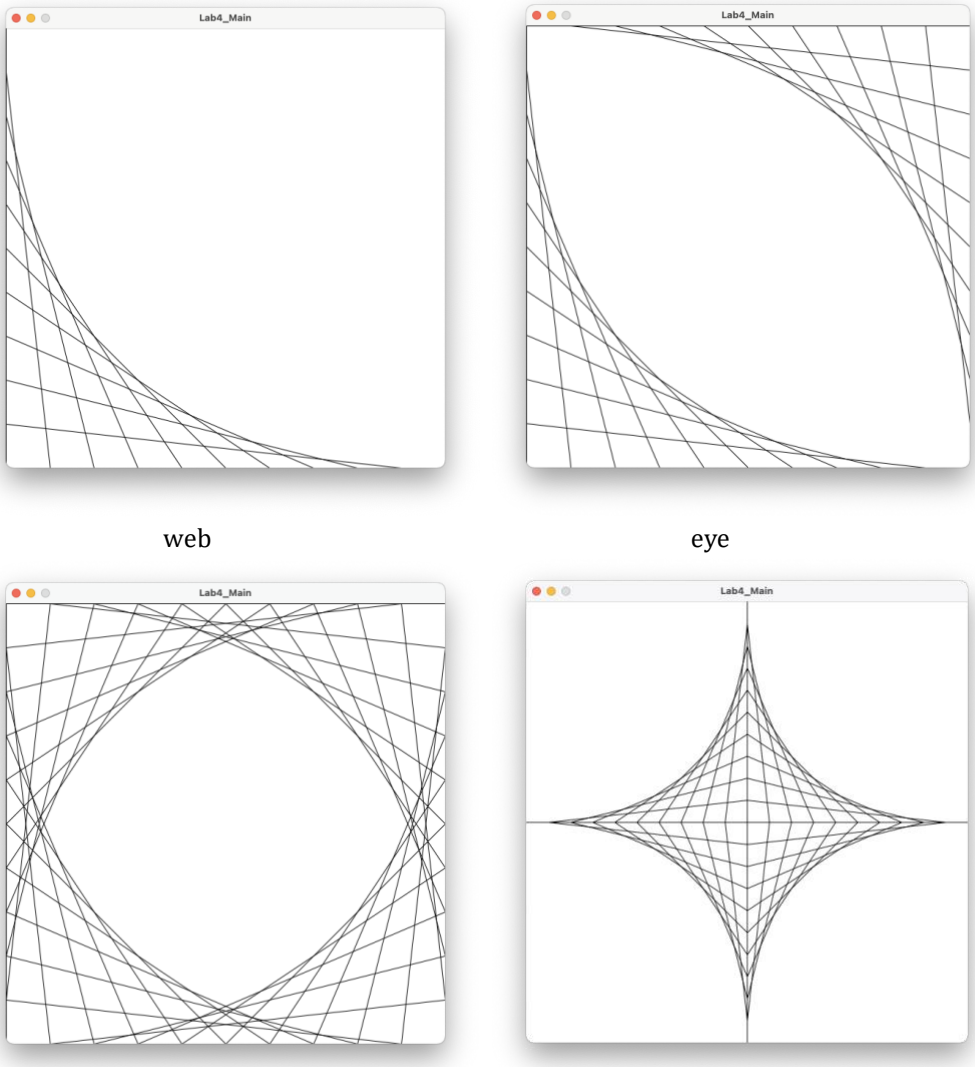


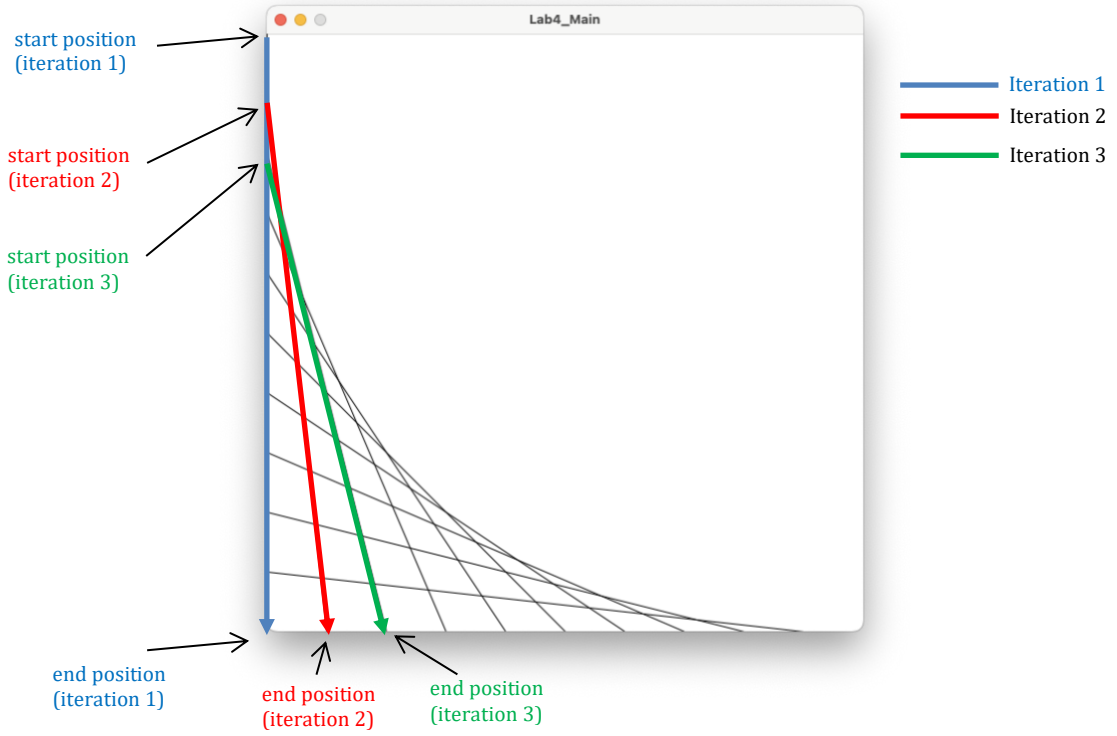
Figure 1: basic line/string art patterns

You may recognize this from some physical crafts you have seen/made as a kid, where you have a board with a set of nails on the edges, that you link up with string. Here is a url to give you some ideas about what can be achieved using these simple principles [fancy string art examples].

Part (a):

Create a void method called **`drawStraightCurveWeb()`** that has a single integer argument (`steps`). The method should use a single loop to create the top left pattern in the above figure. To do this, you want to create a for loop with a standard counter

(i) that runs for `steps` or `steps+1` number of iterations. Within your loop, you want to draw a single line (from a *start* position, to an *end* position):



Part (b):

Create a copy of the method in part (a) and rename it to **`drawStraightCurveEye()`**. Modify the method to draw two lines (within the same loop). This method also accepts a single integer argument (`steps`), which controls how many iterations you will have in your loop (i.e. how many times you draw 2 lines). Your goal is to achieve the "eye" pattern show in the top right of Figure 1.

Part (c):

In parts (a) and (b), you should use the entire size of the window (`0,0` to `width,height`). You may consider trying to parametrize these (so they can be repositioned and scaled to a particular location in the app window).

In this task, based on your **`drawStraightCurveWeb()`** and **`drawStraightCurveEye()`**, create your own String-Art artwork. For inspiration, you are shown two possible patterns in the bottom half of Figure 1 (i.e. square/funnel or cross). You can simply try to create one of these (with some colour/ stroke/ strokeWeight variations) or try to generate some other interesting, related pattern (e.g. make a pattern between two arbitrarily positioned/oriented straight lines – as opposed to the screen edge).

*** you should not need more than one loop for Question 1 (in any of these methods).*

Question02 (practice working with basic arrays)

Emulate/run some of the example code from the lectures (i.e. *Week 6: L11 & L12*, and *W6_ArraysAndLoops.zip* → e.g. finding the sum/max/min elements in the array, etc.) Then attempt the following exercises by following the steps outlined below, and completing the associated method definitions within **`Question02.pde`**.

In the **`question2()`** method (located inside `Question2.pde`), an integer array (**`intSamples`**) of size `15` has been declared and initialized to hold a set of integers for you to use as an argument for some of the tasks below. Complete the methods (also located inside `Question2.pde`), for each of the following operations, where each of the methods assumes the provided *method signature*.

Test your methods by running them from the **`question2()`** method (passing appropriate arguments / assigning return values to appropriate variables as needed). To run **`question2()`**, uncomment the statement that calls this method within your **`lab4.pde`** file (you may comment out any calls relating to previous questions).

1. Complete the method: **`showArray(…)`** below that outputs (i.e. prints) the values from the `intSamples` array to the screen. This method should receive an integer array argument (`array`), along with name of the array (`arrayName`), and has no return value. Use this method to print out `intSamples` (generated above).

```
void showArray(String arrayName, int[] array) { … }
```

Example output after running (format should follow this exactly):

```
showArray("intSamples", intSamples);
```

```
==========
intSamples:
[ 154, 12, 48, 174, 85, 251, 114, 164, 135, 88, 297, 9,  51, 95, 241 ]
==========
```

2. Find and return the <u>average</u> value of the elements in the array (i.e. the sum of all the values divided by the number of elements in the array) :

```
float average(int[] myA) { … }
```

Example output after running (note this method does not print anything itself):

```
println("=> average of intSamples = " + nf(average(intSamples),2,3));
```

```
=> average of intSamples = 127.867
```

3. Find the range of an array of integers. The range is the difference between the maximum and minimum value in an array. First you need to find the maximum and minimum, then compute and return their difference. As the input array (`myA`) is an integer array, the output should also be an integer.

```
int range(int[] myA) { … }
```

Example output after running

```
println("=> range of intSamples = " + range(intSamples));
```

```
=> range of intSamples = 288
```

4. Calculate the number of elements that fall within the range defined by integer arguments (`min`) and (`max`), you can assume `max>min` always.

```
int sumValuesInRange(int min, int max, int[] myA) { … }
```

Example output after running

```
println("=> sum of intSamples within (110,200) = " +
                              sumValuesInRange(110,200,intSamples) );
println("=> sum of intSamples within (50,150) = " +
                              sumValuesInRange(50,150,intSamples) );
```

```
=> sum of intSamples within (110,200) = 741
=> sum of intSamples within (50,150) = 568
```

5. Write a method that shuffles the elements in an array (returning a new `int[]` array from an existing `int[]` array then shuffling the elements around (i.e. changing the order of elements in the array).

```
int[] shuffleElements(int[] myArray) { … }
```

One possible algorithm to do this is to:

- create a new array the same size as the argument array (`myArray`)
- do a fixed number of iterations (usually more than there are elements in the array), and for each iteration:
    - pick a random element, and swap it with the first element

To pick a random element, use the `random()` method available within processing to select a random index over any of the indexes other than 0 (remember, 0 is the index for the first element).

When you do a swap, you need to make sure you don't overwrite and lose one of the two elements being swapped. To do this:

- save one of the elements (e.g. first element in array) into a third variable
- copy the random element (the one from a random index between the second and last element) into the first location of the array
- then write the saved element originally at the first location back into the location you got the random element from.

Example output after running:

```
int[] intSamplesShuffled = shuffleElements(intSamples);
showArray("intSamplesShuffled", intSamplesShuffled);
```

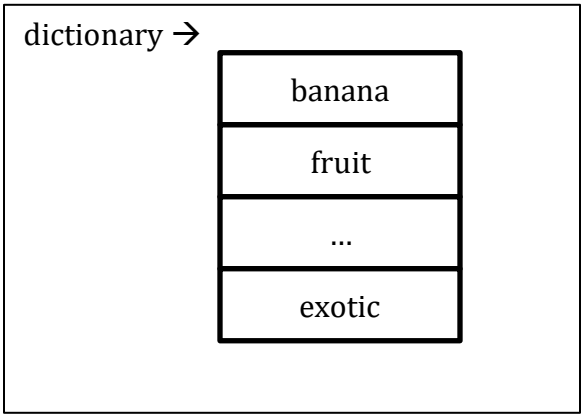
```
==========
intSamplesShuffled =
[ 48, 85, 12, 174, 135, 251, 114, 88, 297, 51, 9, 164, 241, 95, 154 ]
==========
```

Question 3: Create a simple Hangman Game (using String & char arrays)

Write a program that creates a dictionary of words as a String array. Initialize the array in the **`question3()`** method by creating a String of space separated words, and tokenizing it into a `String[]` array using the **`split()`** method from processing.

Make this dictionary contain ~20 (words) – add any words you wish (preferrably greater than 5 characters per word – to make things more interesting).

Here's the general idea:



When the program is run, the dictionary (tokenized String array) is initialized, then a word is randomly selected from the dictionary. Display the word as a "hidden word" to the screen (either on the console or as text in the app window):

```
_ _ _ _ _ _
```

For example, if the randomly selected word is "banana", six underscore (masked) characters ("_") appear, as above.

The program runs via setup() & draw() and waits for the user to guess a letter (by typing a keypress). The number of guesses (keypresses) is recorded as the game continues.

For each guess, check all places where the letter is in the word. If so, output the hidden word again, revealing the correctly-guessed letters. If the guess fails, indicate this with an increase in guess count, but no change in displaying the output (you have some creative freedom in how you show/indicate this).

For example, if the user presses "a", the following might appear:

```
_ a _ a _ a
```

If the letter is NOT in the hidden word, print a message saying so, then output the hidden word again. Continue until all the letters of the word are revealed.

```
_ a _ a _ a

(sorry, that letter doesn't exist)
```

```
b a _ a _ a
```

...

```
b a n a n a

(Correct, you guessed it!)
```

Hints:

- You will need a series of loops and likely more than one `char` array or `String` to achieve this task
- Consider converting the word selected from the dictionary from a `String` to a `char[]` array and working with that
- Consider making a duplicate `char[]` array to track hidden/revealed characters.. until they are know you can show `'-'` or `'*'` as a mask for each character in the word. Then as guesses are made, you can reveal those characters.. until there are no masking characters left.
- You can use a for loop to check if a character exists in a `char` array or `s`tring, there are also methods in the `String` type that can help you do this
- You may output the status of the guess and any messages to the console (using `print` methods), or to the app window (using `text` methods)

STEP 3: SUBMISSION (Deadline 5:00pm Tue Nov 8th, 2022)

NOTE: ***REMEMBER,*** *you can choose to use the web-submit function (see Lab 0 for walkthrough). You will need to find and upload your lab4 *.pde files independently if doing it this way.*

Web-submit can also be used to check your submission from the terminal, and can be found at the link: https://webapp.eecs.yorku.ca/submit/