# CS106 W20 - Assignment 07

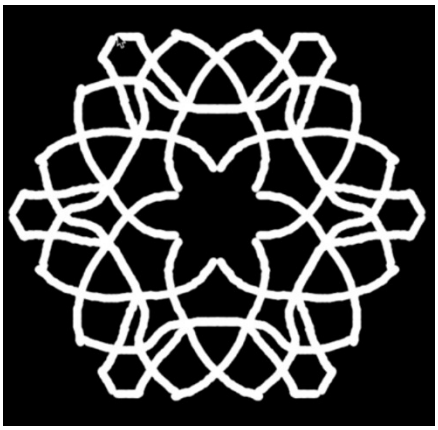
# Geometric Context

Due: Friday, March 6, 11:59 PM

---

## Question 1: Kaleidoscope [5 marks]

The difference between an unmotivated doodle and a beautiful snowflake is just six turns and a flip. See the Lecture06 handout example *Snowflake* for a setup that draws with exactly these symmetries. Where it draws random triangles, here, you will draw your user's mouse drags.



https://www.youtube.com/watch?v=AXI1pggY0SI

In the provided starter code, open the sketch KaleidoDraw. You will see a very simple, self-contained drawing tool. Every mouse drag draws a line in the sketch window. Pressing the space bar clears the drawing and pressing the 's' key saves it.

Modify the sketch so that it lets you draw with the six-fold rotational and reflected symmetries of a snowflake. That is, the drawing should look the same if you rotate it around the centre of the sketch window by 60 degrees, and one side should be a mirror image of the other side. The video above shows what the sketch should look like when you're done.

To make this work, you must change the contents of the mouseDragged() function, and optionally add helper functions. Instead of drawing a single line, as in the starter, draw *twelve* lines in which:

- [1 mark] the shape of the mouse-dragged line is seen
- [1 mark] rotated six times around the centre of the sketch window, and also
- [1 mark] reflected across an appropriate line through the centre.
- [1 mark] You should preserve the point of user interactivity in which one of these lines is directly under the dragged mouse.
- [1 mark] In a purely context-based solution, every call to line() is identical to the one that comes in the initial sketch.

It is possible to do all of this by adding fewer than a dozen lines of code to the starter.

Save your sketch as KaleidoDraw in your A07 folder.

## Question 2: Hierarchical drawing [7 marks]

Write a Processing sketch that draws a scene of your invention, making use of geometric context functions. The scene can consist of anything you want, as long as it satisfies the conditions below.

1. Your sketch window should have size at least 400×400, and no more than 1000×1000. Choose a size that's most appropriate for your scene.
2. [2 marks] The scene should be *figurative*: it should contain clear objects (people, animals, plants, buildings, vehicles, etc.), and not consist entirely of abstract shapes.
3. Don't use any data from external files (images, SVG files, text files, etc.). All drawing should be based on calls to built-in drawing functions in the sketch itself.
4. [1 mark] At least one object must be an "advanced shape" (i.e., it must be drawn using beginShape(), vertex(), and endShape()).
5. [2 marks] There must be at least one object that's *repeated*: it should appear at least three times with different transformations, carried out using geometric context. Put the code to draw the object into a helper function, and call that function three times wrapped inside different transformations. At least one of the copies should be scaled, and at least one should be rotated.

   Note that "repetition" doesn't necessarily imply the use of a loop. Suppose you have a helper function called myShape(). Then both of the following would be considered to contain three calls to the function:

```
push();
// Perform some transformations
myShape();
pop();

push();
// Perform some transformations
myShape();
```

```
pop();

push();
// Perform some transformations
myShape();
pop();
```

```
for (let i = 0; i < 3; i++) {
  push();
  // Perform some transformations
  myShape();
  pop();
}
```

6. [2 marks] At least one object must use more than one level of hierarchy: a geometric context nested inside of another geometric context. You need to have at least one place where a call to push() occurs while another context has *already* been pushed. See, for example, the door() and house() functions in the HierarchicalStreet sample sketch to see places where nested contexts are used.

The goal of this question is to practice using translate(), rotate(), scale(), push() and pop(), but in an open-ended way. You should think carefully about what sort of scene you'd like to draw within the constraints above. We encourage you to design a visually rich and interesting scene. We'll award bonus marks for especially creative or artistic results.

Save your work in a sketch called HierarchicalScene, inside your A07 folder.

## General Correctness

- One mark will be deducted for files or directories named incorrectly (the zip file, etc.)
- One or more marks will be deducted if the program crashes (depending on the severity).
- Few or no marks will be awarded for solutions that use manual arithmetic or extra parameters, where a geometric transformation is required.

Assignments that do not run may receive a grade of 0. Even if you don't complete the entire assignment, don't leave it in a broken state. Make sure it runs so we can find ways to give you part marks.

## Coding Style and Efficiency [2 marks]

Follow the course coding style for whitespace and comments. Consult the "Code Style Guide" on LEARN. For example:

- Comment your code appropriately. Avoid superfluous comments.
- Correctly and consistently indent your code blocks.
- Use correct inline spacing for variable declaration and assignment.
- Use good line spacing to chunk sections of your code.
- Pay special attention to inline spacing for your conditional statements.
- Use semicolons.
- Use let or const, never var.
- All variables must be declared using let or const. Don't use variables that have not been declared.

One or more marks may be deducted for solutions that have obvious inefficiencies.

- Variables that are declared or assigned, but not used.
- Unnecessarily variables that are duplicates of other variables.
- Unnecessarily repeating the same code in multiple places.
- Too many "magic numbers": the same number appears in many places indicating a variable should have been used instead.

**Submission**

When you are ready to submit, please follow these steps.

1. Please ensure that any sketches you submit compile and run. It's better to submit a sketch that runs smoothly but implements fewer required features than one that has broken code for all features. If you get partway into a feature but can't make it work, comment it out so that the sketch works correctly without it.
2. If necessary, review the Code Style Guide and use Processing's built-in auto format tool. You do not need to use the precise coding style outlined in the guide, but whatever style you use, your code must be clear, concise, consistent, and commented.
3. If necessary, review the How To Submit document for a reminder on how to submit to LEARN.
4. Make sure to include a comment at the top of all source files containing your name and student ID number.
5. Create a zip file called A03.zip containing the entire A03 with its subfolders Stars and Emissions.
6. Upload A03.zip to LEARN. Remember that you can (and should!) submit as many times as you like. That way, if there's a catastrophe, you and the course staff will still have access to a recent version of your code.
7. If LEARN isn't working, and ***only*** if LEARN isn't working, please email your ZIP file to the course account (see the course home page for the address). In this case, you *must mail your ZIP file before the deadline.* Please use this only for emergencies, not "just in case". Submissions received after the deadline may receive feedback, but their marks will not count.