All my final code

365 hexagons

final int hexSize = 24; // radius of each hexagon (distance from center to any corner)

final int rows = 14; // number of hexagons per row (representing a fortnight)

final int columns = 26; // number of hexagons per column (14 fortnights)

// set up a larger canvas size to fit all hexagons properly

// canvas width adjusted to include spacing between hexagons and make sure all hexagons fit properly

final int canvasWidth = (int)((rows + 1) \* (2 \* hexSize \* 1.5)) + 2000 + 30; // add 30 pixels for a bit more width

// canvas height adjusted to include spacing between hexagons and ensure all hexagons fit

final int canvasHeight = (int)(columns \* (sqrt(3) \* hexSize \* 1.5)) + 300;

void settings() {

size(canvasWidth, canvasHeight); // set up the canvas size here, using settings()

}

void setup() {

background(255); // set the background color to white

save("final\_365\_hexagons.png");

// load the JSON file containing temperature data from Visual Crossing Weather API

JSONObject jsonData = loadJSONObject("sydney\_temperature\_2023.json");

// get the array of daily data from the JSON object

JSONArray days = jsonData.getJSONArray("days");

// set up drawing parameters for positioning hexagons

int xOffset = 40; // x-axis start offset to provide some padding

int yOffset = 50; // y-axis start offset to provide some padding

float horizontalSpacing = hexSize \* 2 \* 1.5; // horizontal spacing between hexagons to ensure they don't overlap

float verticalSpacing = sqrt(3) \* hexSize \* 1.5; // vertical spacing between hexagons to ensure they don't overlap

// loop through the data and draw hexagons for each day in the dataset

for (int i = 0; i < days.size(); i++) {

JSONObject day = days.getJSONObject(i);

String date = day.getString("datetime");

float high = day.getFloat("tempmax");

float low = day.getFloat("tempmin");

float avg = (high + low) / 2.0; // calculate the average temperature for the day (using 2.0 to ensure float division)

// map temperatures to gradient colours specific to each day for the border (low temperature)

// gradient colours help visualize the temperature change; the mapping is inspired by a typical colour gradient

// ranging from cool (pink, blue) to warm (orange, red)

color borderColor;

if (low <= 5) {

borderColor = lerpColor(color(255, 182, 193), color(255, 0, 255), map(low, 0, 5, 0, 1)); // Baby Pink to Fuchsia for very low temperatures

} else if (low <= 10) {

borderColor = lerpColor(color(148, 0, 211), color(75, 0, 130), map(low, 5, 10, 0, 1)); // Violet to Indigo for low temperatures

} else if (low <= 15) {

borderColor = lerpColor(color(0, 0, 255), color(144, 238, 144), map(low, 10, 15, 0, 1)); // Blue to Light Green for moderate temperatures

} else if (low <= 20) {

borderColor = lerpColor(color(144, 238, 144), color(255, 255, 0), map(low, 15, 20, 0, 1)); // Light Green to Yellow for warm temperatures

} else if (low <= 25) {

borderColor = lerpColor(color(255, 255, 0), color(255, 127, 0), map(low, 20, 25, 0, 1)); // Yellow to Orange for warmer temperatures

} else if (low <= 30) {

borderColor = lerpColor(color(255, 127, 0), color(255, 0, 0), map(low, 25, 30, 0, 1)); // Orange to Red for hot temperatures

} else {

borderColor = lerpColor(color(255, 0, 0), color(139, 0, 0), map(low, 30, 40, 0, 1)); // Red to Dark Red for very hot temperatures

}

color middleColor = color(255); // Set the middle color to white for now (Average Temperature)

// determine center colour based on the high temperature for the day

// this gives a visual indicator for hotter or colder days!

//inspired by original temperature blanket designs, which are linked on Github

color centerColor;

if (high <= 7.5) {

centerColor = lerpColor(color(255, 182, 193), color(255, 0, 255), map(high, 0, 7.5, 0, 1)); // Baby Pink to Fuchsia for very low temperatures

} else if (high <= 15) {

centerColor = lerpColor(color(148, 0, 211), color(75, 0, 130), map(high, 7.5, 15, 0, 1)); // Violet to Indigo for low temperatures

} else if (high <= 20) {

centerColor = lerpColor(color(0, 0, 255), color(144, 238, 144), map(high, 15, 20, 0, 1)); // Blue to Light Green for moderate temperatures

} else if (high <= 22.5) {

centerColor = lerpColor(color(144, 238, 144), color(255, 255, 0), map(high, 20, 22.5, 0, 1)); // Light Green to Yellow for warm temperatures

} else if (high <= 25) {

centerColor = lerpColor(color(255, 255, 0), color(255, 127, 0), map(high, 22.5, 25, 0, 1)); // Yellow to Orange for warmer temperatures

} else if (high <= 30) {

centerColor = lerpColor(color(255, 127, 0), color(255, 0, 0), map(high, 25, 30, 0, 1)); // Orange to Red for hot temperatures

} else {

centerColor = lerpColor(color(255, 0, 0), color(139, 0, 0), map(high, 30, 40, 0, 1)); // Red to Dark Red for very hot temperatures

}

// calculate x and y positions for each hexagon

int row = i / rows; // determine which row the hexagon belongs to

int col = i % rows; // determine the column for the hexagon

float x = xOffset + row \* horizontalSpacing; // calculate x position based on row and spacing

float y = yOffset + col \* verticalSpacing; // calculate y position based on column and spacing

// Draw the hexagon layers

drawHexagon(x, y, hexSize, borderColor, middleColor, centerColor);

}

}

void drawHexagon(float x, float y, float radius, color bordercolor, color middlecolor, color centercolor) {

// draw the border hexagon (outer layer)

stroke(bordercolor); // set border colour based on temperature

strokeWeight(1);

fill(bordercolor, 191); // set fill colour with 75% opacity for the border color

drawSingleHexagon(x, y, radius); // draw outer layer hexagon

// draw the middle hexagon (middle layer, slightly smaller)

stroke(middlecolor); // set stroke colour to middle colour

strokeWeight(1);

fill(middlecolor);

drawSingleHexagon(x, y, radius \* 0.7); // draw middle layer with reduced radius for a nested effect

// draw the center hexagon (inner layer)

stroke(centercolor); // set stroke colour to center colour based on highest temperature

strokeWeight(1);

fill(centercolor, 200); // slight opacity for the center colour to emphasize it

drawSingleHexagon(x, y, radius \* 0.5); // draw inner hexagon with smaller radius for emphasis

}

void drawSingleHexagon(float x, float y, float radius) {

beginShape();

for (int i = 0; i < 6; i++) {

float angle = PI / 3 \* i; // adjust angle for flat edge alignment

float xOffset = cos(angle) \* radius; // calculate x offset for vertex

float yOffset = sin(angle) \* radius; // calculate y offset for vertex

vertex(x + xOffset, y + yOffset); // set vertex point

}

endShape(CLOSE); // close the hexagon shape

}

//done!!!

visualisation of all 24 hexagons

//hexagon radius reduced to 100 to make sure the hexagons fit comfortably.

//border thickness: adjusted to 10 for visual balance.

//canvas and spacing adjustments: horizontal and vertical spacing adjusted to ensure there is enough room between the hexagons and they don't overlap.

import processing.embroider.\*;

import processing.data.JSONObject;

import processing.data.JSONArray;

PEmbroiderGraphics E;

float hexRadius = 100; // reduced radius of hexagon (distance from center to any corner)

float borderThickness = 10; // adjusted thickness of the border hexagon

JSONObject temperatureData;

void setup() {

size(2300, 2400); // canvas size for 24 hexagons (6 columns x 4 rows)

E = new PEmbroiderGraphics(this, width, height);

String outputFilePath = sketchPath("Full\_Year\_Visualization.vp3");

String pesOutputFilePath = sketchPath("Full\_Year\_Visualization.pes");

E.setPath(outputFilePath);

E.setPath(pesOutputFilePath); // save as PES file with the correct name

// upload the temperature data from JSON file

temperatureData = loadJSONObject("sydney\_temperature\_2023.json");

E.beginDraw(); // start drawing the embroidery; sourced from PEmbroider library

E.clear(); // clear any previous content from the embroidery canvas

// draw the hexagons with temperatures from temperature data

drawAllMonthlyHexagons();

E.optimize(); // slow but necessary for good output

E.endDraw(); // write the embroidery file

E.visualize(); // visualize the embroidery stitches

}

// function to draw all monthly hexagons for the full year

void drawAllMonthlyHexagons() {

String[] months = {"2023-01", "2023-02", "2023-03", "2023-04", "2023-05", "2023-06",

"2023-07", "2023-08", "2023-09", "2023-10", "2023-11", "2023-12"};

int counter = 0;

for (int row = 0; row < 2; row++) { // iterate through all 4 rows

for (int col = 0; col < 6; col++) { // iterate through 6 columns

String month = months[counter];

float cx = 200 + col \* 300; // adjusted horizontal spacing between columns

float cy = 200 + row \* 500; // increased spacing to fit 4 rows properly

drawMonthlyHexagons(month, cx, cy);

counter++;

}

}

}

// function to draw hexagons for a specific month

void drawMonthlyHexagons(String monthPrefix, float cx, float cy) {

// extract temperature data from JSON file

JSONArray days = temperatureData.getJSONArray("days");

float extremeLow = Float.MAX\_VALUE;

float extremeHigh = Float.MIN\_VALUE;

float sumHigh = 0;

float sumLow = 0;

int count = 0;

// repeat over all days in the JSON array to find temperatures for the specified month

for (int i = 0; i < days.size(); i++) {

JSONObject day = days.getJSONObject(i);

String date = day.getString("datetime");

if (date.startsWith(monthPrefix)) {

float tempMax = day.getFloat("tempmax"); // get maximum temperature of the day

float tempMin = day.getFloat("tempmin"); // get minimum temperature of the day

if (tempMax > extremeHigh) extremeHigh = tempMax; // update extreme high if needed

if (tempMin < extremeLow) extremeLow = tempMin; // update extreme low if needed

sumHigh += tempMax; // add to sum of daily highs

sumLow += tempMin; // add to sum of daily lows

count++; // increase day count

}

}

float averageHigh = sumHigh / count; // calculate average high temperature for the month

float averageLow = sumLow / count; // calculate average low temperature for the month

// draw the hexagons for the given month (extremes and averages)

drawHexagonWithBorder(cx, cy, getSeason(monthPrefix), "extreme", extremeLow, extremeHigh); // extreme low and high

drawHexagonWithBorder(cx, cy + 225, getSeason(monthPrefix), "average", averageLow, averageHigh); // average low and high

}

// function to determine the season based on the month

String getSeason(String month) {

switch(month) {

case "2023-03":

case "2023-04":

case "2023-05":

return "Autumn";

case "2023-06":

case "2023-07":

case "2023-08":

return "Winter";

case "2023-09":

case "2023-10":

case "2023-11":

return "Spring";

case "2023-12":

case "2023-01":

case "2023-02":

return "Summer";

default:

return "";

}

}

// function to draw a hexagon with a border, using the season and temperature data for styling

void drawHexagonWithBorder(float cx, float cy, String season, String type, float temperatureLow, float temperatureHigh) {

// draw the border hexagon

E.noFill();

E.stroke(0);

E.strokeWeight(borderThickness);

E.strokeMode(PEmbroiderGraphics.PERPENDICULAR);

E.strokeSpacing(4);

if (season.equals("Autumn")) {

// running stitch in a wavy pattern for autumn border

E.setStitch(8, 16, 0.5);

E.hatchMode(PEmbroiderGraphics.PARALLEL);

E.hatchAngleDeg(30); // angled offset for a falling leaves impression

} else if (season.equals("Winter")) {

// cross-stitch or straight stitch with snowflake-like details for winter border

E.setStitch(6, 12, 0.2);

E.hatchMode(PEmbroiderGraphics.PERPENDICULAR);

} else if (season.equals("Spring")) {

// light satin border with a simple floral motif for spring

E.setStitch(4, 8, 0.3);

E.hatchMode(PEmbroiderGraphics.SATIN);

} else if (season.equals("Summer")) {

// thicker zigzag stitch border, reminiscent of sun rays for summer

E.setStitch(10, 20, 0.4);

E.hatchMode(PEmbroiderGraphics.ZIGZAG);

}

drawHexagon(cx, cy, hexRadius + borderThickness / 2); // draw border hexagon

// set color based on temperature for the inner hexagon

float temperature = temperatureHigh;

if (temperature <= 7.5) {

E.fill(810); // light lilac

} else if (temperature <= 10) {

E.fill(620); // magenta

} else if (temperature <= 12.5) {

E.fill(614); // purple

} else if (temperature <= 15) {

E.fill(405); // blue

} else if (temperature <= 17.5) {

E.fill(19); // sky blue

} else if (temperature <= 20) {

E.fill(534); // teal green

} else if (temperature <= 22.5) {

E.fill(509); // leaf green

} else if (temperature <= 25) {

E.fill(513); // lime green

} else if (temperature <= 27.5) {

E.fill(202); // lemon yellow

} else if (temperature <= 30) {

E.fill(208); // orange

} else if (temperature <= 35) {

E.fill(126); // pumpkin

} else if (temperature <= 37.5) {

E.fill(800); // red

} else {

E.fill(807); // carmine

}

// draw the inner hexagon (center fill)

E.noStroke();

if (season.equals("Autumn")) {

// parallel hatch with an angled offset for a falling leaves impression

E.setStitch(5, 10, 0.5);

E.hatchMode(PEmbroiderGraphics.PARALLEL);

E.hatchAngleDeg(30);

} else if (season.equals("Winter")) {

// spiral hatch for a swirling snowflake effect

E.setStitch(5, 15, 0.3);

E.hatchMode(PEmbroiderGraphics.SPIRAL);

} else if (season.equals("Spring")) {

// concentric hatch for a blooming effect

E.setStitch(4, 12, 0.4);

E.hatchMode(PEmbroiderGraphics.CONCENTRIC);

} else if (season.equals("Summer")) {

// satin hatch for a vibrant, smooth effect

E.setStitch(6, 14, 0.3);

E.hatchMode(PEmbroiderGraphics.SATIN);

}

drawHexagon(cx, cy, hexRadius - borderThickness); // draw inner hexagon

}

// function to draw a hexagon using trigonometric calculations for vertices

void drawHexagon(float cx, float cy, float radius) {

float angleOffset = PI / 6; // offset to ensure flat sides are facing each other

E.beginShape();

for (int i = 0; i < 6; i++) {

float angle = angleOffset + i \* PI / 3;

float x = cx + cos(angle) \* radius;

float y = cy + sin(angle) \* radius;

E.vertex(x, y); // add vertex at calculated coordinates

}

E.endShape(CLOSE); // complete the hexagon shape

}

import processing.embroider.\*;

import processing.data.JSONObject;

import processing.data.JSONArray;

PEmbroiderGraphics E;

float hexRadius = 200; // adjust the radius of hexagon (distance from center to any corner)

float borderThickness = 20; // adjust the thickness of the border hexagon

JSONObject temperatureData;

void setup() {

size(1000, 1000); // this adjusts the canvas size to match the 4x4 inch embroidery hoop on the Brother SE425 Embroidery Machine

E = new PEmbroiderGraphics(this, width, height);

String outputFilePath = sketchPath("Final\_Jan\_and\_Feb.vp3"); //saves as a VP3 file

String pesOutputFilePath = sketchPath("Final\_Jan\_and\_Feb.pes"); //saves as a PES file, which is what is uploaded into the SE425

E.setPath(outputFilePath);

E.setPath(pesOutputFilePath); // save as PES file with the correct name

// upload the temperature data from JSON file

temperatureData = loadJSONObject("sydney\_temperature\_2023.json"); // data sourced from Visual Crossing Weather API

E.beginDraw(); //sourced from PEmbroider library)

E.clear();

// draw the hexagons with temperatures from temperature data

drawMonthlyHexagons("2023-01", 250, 250); // january (top left hexagon) (two hexagons stacked vertically on top of each other)

drawMonthlyHexagons("2023-02", 700, 250); // february (top right hexagon) (two hexagons stacked vertically on top of each other)

// sourced from PEmbroider library

E.optimize(); // slow but necessary for good output. if there is too much code then removing this line can make it run faster.

E.endDraw(); // write the embroidery file

E.visualize(); //too see the embroidery stitches and how they are arranged

}

// function which is used to draw hexagons with different temperature values for a given month in 2023

// method for temperature extraction and averaging based on Processing.org

void drawMonthlyHexagons(String monthPrefix, float cx, float cy) {

// extract temperature data from JSON file

JSONArray days = temperatureData.getJSONArray("days"); // extract the "days" array from the JSON object containing temperature data

float extremeLow = Float.MAX\_VALUE; // used to find the lowest temperature of the month

float extremeHigh = Float.MIN\_VALUE; // used to find the highest temperature of the month

float sumHigh = 0; // variable to combine the daily high temperatures for calculating the average highest temperature of the month

float sumLow = 0; // variable to combine the daily low temperatures for calculating the average lowest temperature of the month

int count = 0; // this is a counter to keep a record of the number of days processed in the month

// repeat over all days in the JSON array to find temperatures for the specified month that i want

for (int i = 0; i < days.size(); i++) {

JSONObject day = days.getJSONObject(i); // extract the current day’s JSON object

String date = day.getString("datetime"); // get the date string in a "yyyy-MM-dd" format (another reminder im glad to not be in america)

if (date.startsWith(monthPrefix)) { // check if the date actually belongs to the given month

float tempMax = day.getFloat("tempmax"); // find out the maximum temperature for the day

float tempMin = day.getFloat("tempmin"); // find out the minimum temperature for the day

if (tempMax > extremeHigh) extremeHigh = tempMax; // update extremeHigh if the current day's maximum temperature is higher (conditional)

if (tempMin < extremeLow) extremeLow = tempMin; // update extremeLow if the current day's minimum temperature is lower (conditional)

sumHigh += tempMax; // add the current day's maximum highest temperature to the total sum for averaging

sumLow += tempMin; // add the current day's minimum lowest temperature to the total sum for averaging

count++; // increase the counter to keep track of how many days are processed

}

}

float averageHigh = sumHigh / count; // calculate the average high temperature for the month

float averageLow = sumLow / count; // calculate the average low temperature for the month

// draw the hexagons for the given month (extremes and averages)

drawHexagonWithBorder(cx, cy, getSeason(monthPrefix), "extreme", extremeLow, extremeHigh); // extreme low and high

drawHexagonWithBorder(cx, cy + 450, getSeason(monthPrefix), "average", averageLow, averageHigh); // average low and high

}

// this is a function that determines the season based on what month of the year it is

// I just assigned month-to-season mapping based on what normally happens in southern hemisphere

String getSeason(String month) {

switch(month) {

case "2023-03":

case "2023-04":

case "2023-05":

return "Autumn";

case "2023-06":

case "2023-07":

case "2023-08":

return "Winter";

case "2023-09":

case "2023-10":

case "2023-11":

return "Spring";

case "2023-12":

case "2023-01":

case "2023-02":

return "Summer";

default:

return "";

}

}

// this is a function that draws a hexagon with a border around it, with special stitch styling based on the season

// the stitch styles and border patterns are based on source code found in the PEmbroidery library- i have just altered and manipulated them to my own design choices

void drawHexagonWithBorder(float cx, float cy, String season, String type, float temperatureLow, float temperatureHigh) {

// draw the border hexagon which has the lower temperatures (cant overlap)

E.noFill();

E.stroke(0);

E.strokeWeight(borderThickness);

E.strokeMode(PEmbroiderGraphics.PERPENDICULAR);

E.strokeSpacing(4);

if (season.equals("Autumn")) {

// running stitch in a wavy pattern for autumn border

E.setStitch(8, 16, 0.5);

E.hatchMode(PEmbroiderGraphics.PARALLEL);

E.hatchAngleDeg(30); // angled offset it kind of looks like falling leaves

} else if (season.equals("Winter")) {

// cross-stitch/straight stitch with snowflake-like details for winter border

E.setStitch(6, 12, 0.2);

E.hatchMode(PEmbroiderGraphics.PERPENDICULAR); // uses PEmbroiderGraphics.PERPENDICULAR

} else if (season.equals("Spring")) {

// light satin border with a cute floral motif for spring vibes

E.setStitch(4, 8, 0.3);

E.hatchMode(PEmbroiderGraphics.SATIN);

} else if (season.equals("Summer")) {

// thicker zigzag stitch border, vaguely reminiscent of sun rays for summer

E.setStitch(10, 20, 0.4);

E.hatchMode(PEmbroiderGraphics.ZIGZAG);

}

drawHexagon(cx, cy, hexRadius + borderThickness / 2);

// set the colour based on temperature for the centre hexagon (using the higher temperature)

// colour-coding for temperature inspired by various crocheted/knitted/hand-embroidered temperature blankets made for sydney (images in GitHub)

// these colours and their values are based on the SimThread 63 Spools Colour Chart (also linked in GitHub)

float temperature = temperatureHigh;

if (temperature <= 7.5) {

E.fill(810); // light lilac

} else if (temperature <= 10) {

E.fill(620); // magenta

} else if (temperature <= 12.5) {

E.fill(614); // purple

} else if (temperature <= 15) {

E.fill(405); // blue

} else if (temperature <= 17.5) {

E.fill(19); // sky blue

} else if (temperature <= 20) {

E.fill(534); // teal green

} else if (temperature <= 22.5) {

E.fill(509); // leaf green

} else if (temperature <= 25) {

E.fill(513); // lime green

} else if (temperature <= 27.5) {

E.fill(202); // lemon yellow

} else if (temperature <= 30) {

E.fill(208); // orange

} else if (temperature <= 35) {

E.fill(126); // pumpkin

} else if (temperature <= 37.5) {

E.fill(800); // red

} else {

E.fill(807); // carmine

}

// draw the inner hexagon (center fill) (needs space away from the border so the machine has room for the complex stitching)

E.noStroke();

// stitch fills are sourced from the PEmbroider library, but their dimensions have been manipulated by me

if (season.equals("Autumn")) {

// parallel hatch with an angled offset for a falling leaves vibes again

E.setStitch(5, 10, 0.5);

E.hatchMode(PEmbroiderGraphics.PARALLEL);

E.hatchAngleDeg(30);

} else if (season.equals("Winter")) {

// spiral hatch which could be interpreted as a swirling snowflake

E.setStitch(5, 15, 0.3);

E.hatchMode(PEmbroiderGraphics.SPIRAL);

} else if (season.equals("Spring")) {

// concentric hatch, reminds me of blooming and flowers

E.setStitch(4, 12, 0.4);

E.hatchMode(PEmbroiderGraphics.CONCENTRIC);

} else if (season.equals("Summer")) {

// satin hatch because its very vibrant, smooth, and the stitches are condensely packed so the colour will pop

E.setStitch(6, 14, 0.3);

E.hatchMode(PEmbroiderGraphics.SATIN);

}

drawHexagon(cx, cy, hexRadius - borderThickness);

}

// this is a function which is used to draw a hexagon, defining the vertices of the hexagon with trigonometric calculations

// based on Processing.org Regular Polygon Examples

void drawHexagon(float cx, float cy, float radius) {

float angleOffset = PI / 6; // offset to ensure flat sides are facing each other AND NOT THE POINTY SIDES

E.beginShape();

for (int i = 0; i < 6; i++) {

float angle = angleOffset + i \* PI / 3;

float x = cx + cos(angle) \* radius;

float y = cy + sin(angle) \* radius;

E.vertex(x, y);

}

E.endShape(CLOSE); // done!!! :D

}

import processing.embroider.\*;

import processing.data.JSONObject;

import processing.data.JSONArray;

PEmbroiderGraphics E;

float hexRadius = 200; // radius of hexagon (distance from center to any corner of the canvas)

float borderThickness = 20; // thickness of the border hexagon

void setup() {

size(1000, 1000); // this canvas size matches the one used for the sets of four hexagons and fits the SE245 embroidery hoop

background(255); // this was grey but i changed it to white for better contrast when it is printed

// originally sourced from the PEmbroider library and manipulated and expanded to fit my design specifications

E = new PEmbroiderGraphics(this, width, height);

String outputFilePath = sketchPath("Print\_Alignment\_Dots.vp3"); // save as VP3 file format

String pesOutputFilePath = sketchPath("Print\_Alignment\_Dots.pes"); // save as PES file format (used by SE425)

E.setPath(outputFilePath);

E.setPath(pesOutputFilePath); // set PES output path

// Code sourced directly from the PEmbroider library to draw any kind of embroidery design

E.beginDraw(); // start drawing the embroidery design

E.clear(); // clear any previous content from the embroidery canvas

// draw alignment dots to be lightly embroidered in order to position the fabric in real life accurately

addAlignmentDots(); // this is a function to add alignment dots

//sourced directly from PEmbroider library

E.optimize(); // slow but essential for high-quality stitch output

E.endDraw(); // end the embroidery drawing process and save to file (also for the png)

E.visualize(); // visualize the embroidery design, showing the arrangement of stitches (but no colour)

// draw the alignment dots on the canvas so i can save it as a printable PNG that is true to size

addAlignmentDotsForPrint(); // this is a function to add alignment dots for printable use (not embroidery)

// save the visual representation of alignment dots as a PNG image file

save("Alignment\_Dots\_Printable.png");

}

// this is a function to add alignment dots for embroidery, built on from code in the PEmbroider library

void addAlignmentDots() {

float dotSize = 5; // set the size of each alignment dot

E.strokeWeight(2); // set the stitch thickness for the dots to make them more visible (need to experiment with machine to see how thick they actually are in real life)

// first hexagon pair - top left (positioned at (250, 250))

float x1 = 250; // x-coordinate for the first pair of hexagons

float y1 = 250; // y-coordinate for the first hexagon

// add four dots around the top hexagon at (x1, y1)

addFourDotsAroundHexagonE(x1, y1);

// add four dots around the bottom hexagon, directly below the first (y1 + 450 spacing)

addFourDotsAroundHexagonE(x1, y1 + 450);

// second hexagon pair - top right (positioned at (700, 250))

float x2 = 700; // x-coordinate for the second pair of hexagons

float y2 = 250; // y-coordinate for the first hexagon in the second pair

// add four dots around the top hexagon at (x2, y2)

addFourDotsAroundHexagonE(x2, y2);

// add four dots around the bottom hexagon, directly below the first (y2 + 450 spacing)

addFourDotsAroundHexagonE(x2, y2 + 450);

}

// function to add alignment dots for printable use (not embroidery)

void addAlignmentDotsForPrint() {

float dotSize = 5; // set the size of each alignment dot for print

// first hexagon pair - top left (positioned at (250, 250))

float x1 = 250; // x-coordinate for the first pair of hexagons

float y1 = 250; // y-coordinate for the first hexagon

// add four dots around the top hexagon at (x1, y1) for printing

addFourDotsAroundHexagonPrint(x1, y1);

// add four dots around the bottom hexagon, directly below the first (y1 + 450 spacing) for printing

addFourDotsAroundHexagonPrint(x1, y1 + 450);

// second hexagon pair - top right (positioned at (700, 250))

float x2 = 700; // x-coordinate for the second pair of hexagons

float y2 = 250; // y-coordinate for the first hexagon in the second pair

// add four dots around the top hexagon at (x2, y2) for printing

addFourDotsAroundHexagonPrint(x2, y2);

// add four dots around the bottom hexagon, directly below the first (y2 + 450 spacing) for printing

addFourDotsAroundHexagonPrint(x2, y2 + 450);

}

// function to add four alignment dots around a hexagon using the PEmbroider library

void addFourDotsAroundHexagonE(float cx, float cy) {

float dotOffset = hexRadius + borderThickness; // the offset distance for placing dots around the hexagon, taking into account radius and border thickness

float dotSize = 5; // the size of each dot for embroidery

// draw dots around the hexagon using PEmbroider (E.ellipse)

// top left dot

E.ellipse(cx - dotOffset, cy - dotOffset, dotSize, dotSize);

// top right dot

E.ellipse(cx + dotOffset, cy - dotOffset, dotSize, dotSize);

// bottom left dot

E.ellipse(cx - dotOffset, cy + dotOffset, dotSize, dotSize);

// bottom right dot

E.ellipse(cx + dotOffset, cy + dotOffset, dotSize, dotSize);

}

// this is a function to add four alignment dots around a hexagon for printing (not embroidery)

void addFourDotsAroundHexagonPrint(float cx, float cy) {

float dotOffset = hexRadius + borderThickness; // offset distance for placing dots around the hexagon, same as in embroidery version

float dotSize = 5; // size of each alignment dot for printing

// set stroke properties for dots to be printed

stroke(0); // black outline for dots

strokeWeight(2); // thickness of the outline

fill(0); // fill the dots with black color

// draw dots around the hexagon using Processing ellipse function (for printing)

// top left dot

ellipse(cx - dotOffset, cy - dotOffset, dotSize, dotSize);

// top right dot

ellipse(cx + dotOffset, cy - dotOffset, dotSize, dotSize);

// bottom left dot

ellipse(cx - dotOffset, cy + dotOffset, dotSize, dotSize);

// bottom right dot

ellipse(cx + dotOffset, cy + dotOffset, dotSize, dotSize);

}

print the temp for the 24 hexagons in python code

import json

# Load the JSON data

with open('sydney\_temperature\_2023.json', 'r') as f:

temperature\_data = json.load(f)

# Function to extract temperature statistics for a given month

def get\_temperature\_stats(month\_prefix):

days = temperature\_data["days"]

extreme\_low = float('inf')

extreme\_high = float('-inf')

sum\_high = 0

sum\_low = 0

count = 0

for day in days:

date = day["datetime"]

if date.startswith(month\_prefix):

temp\_max = day["tempmax"]

temp\_min = day["tempmin"]

if temp\_max > extreme\_high:

extreme\_high = temp\_max

if temp\_min < extreme\_low:

extreme\_low = temp\_min

sum\_high += temp\_max

sum\_low += temp\_min

count += 1

average\_high = sum\_high / count if count > 0 else None

average\_low = sum\_low / count if count > 0 else None

return {

"extreme\_low": extreme\_low,

"extreme\_high": extreme\_high,

"average\_low": average\_low,

"average\_high": average\_high

}

# Get stats for all the months in 2023

january\_stats = get\_temperature\_stats("2023-01")

february\_stats = get\_temperature\_stats("2023-02")

march\_stats = get\_temperature\_stats("2023-03")

april\_stats = get\_temperature\_stats("2023-04")

may\_stats = get\_temperature\_stats("2023-05")

june\_stats = get\_temperature\_stats("2023-06")

july\_stats = get\_temperature\_stats("2023-07")

august\_stats = get\_temperature\_stats("2023-08")

september\_stats = get\_temperature\_stats("2023-09")

october\_stats = get\_temperature\_stats("2023-10")

november\_stats = get\_temperature\_stats("2023-11")

december\_stats = get\_temperature\_stats("2023-12")

print("January Stats:", january\_stats)

print("February Stats:", february\_stats)

print("March Stats:", march\_stats)

print("April Stats:", april\_stats)

print("May Stats:", may\_stats)

print("June Stats:", june\_stats)

print("July Stats:", july\_stats)

print("August Stats:", august\_stats)

print("September Stats:", september\_stats)

print("October Stats:", october\_stats)

print("November Stats:", november\_stats)

print("December Stats:", december\_stats)