EECS 1710
LAB 3 :: Conditional Logic Practice

Prerequisite (labs 0-2) – 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.
This lab will act as a means of forced study for the logic portion of labtest 1 (however there are also practice tests linked on eClass for examples that are in the format of the labtest)

STEP 1: Importing and unzipping the starter code

Download and extract the following lab2 starter code:
http://www.eecs.yorku.ca/course_archive/2022-23/F/1710/labs/lab3/lab3.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 lab3 lab3
```

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

```
submit -l 1710 lab3
```

STEP 2: Exercises

Question 1: [12 marks]

Complete the following 5 exercises from the codingbat website LOGIC_1 puzzles:

twoAsOne() => https://codingbat.com/prob/p113261
sortaSum() => https://codingbat.com/prob/p183071
greenTicket() => https://codingbat.com/prob/p120633
answerCell() => https://codingbat.com/prob/p110973
alarmClock() => https://codingbat.com/prob/p160543

NOTE: if you simply return a value of the appropriate type (as defined by the return type of the method), you will get a list of unit test cases – you can either code your answers directly into the coding bat website (until you pass all or as many of the tests as possible), and then copy the method over to your ***lab3_q1.pde*** file.

Question 2: **[12 marks]**

In this question ***lab3_q2.pde***, we will build a simple help menu (for use in question 3), that will provide instructions to a user on a number of options that are available when pressing different keys on the keyboard. These instructions will be in the form of a help menu, and as individual messages that are printed when the user hits a relevant key. We will add some visual function to this app in question 3, but for question 2, all of the output will be in the console only.

1. Create a method: **`showHelp()`**, that takes no arguments and returns no arguments, but displays the following, formatted message in the console window. Test your method by calling it from setup() – format the output so that it looks something like the following:

```
Drawing Instructions:
=======================
Click and hold mouse button down to draw in window
Release mouse button to stop drawing
press 'c' to clear the drawing
press 'm' to toggle mirrored mode
press 'h' to show this message
```

2. As discussed in lectures, a dynamic sketch usually involves the **`setup()`** + **`draw()`** methods and/or some additional methods that can capture and process specific "events" when/if they occur.

   Mouse-movements, mouse-button clicks and key-presses (on the keyboard) are examples of common "events" that we may wish to process. A screen-refresh is also an event that occurs regularly (it happens automatically - generally 60x per second).

   Each event is usually associated with a specific method that is called, which processing recognizes as **the** method to call when that associated event occurs. For example:
   - **`void mousePressed() {}`** is the method that processing will automatically call whenever one of the buttons on the mouse is pressed down.
   - **`void keyPressed() {}`** is the method that processing will automatically call whenever one of the keys on the keyboard is pressed

   Likewise, **`void draw() {}`** is the method that will be called every time there is a screen refresh event , and **`void setup() {}`** will be called when the sketch starts (think of sketch as having a start event that calls the setup method).

   Note: *All java programs have a starting point (if* ***`setup()`*** *is not in your program, it just starts from the first statement.. however if any methods are defined in your program, it will look for* ***`setup()`*** *to start everything from. If any of these methods are not in your program, they simply won't be called. So leaving out the* ***`draw()`*** *method for example, just means nothing will happen automatically when the screen refresh event occurs.*

   In this step, you are asked to modify the method for processing key presses, so that each of the possibilities reflected in the **`showHelp()`** menu are displayed:

The keyword **`key`** contains the value (a character) of the last key pressed, and can be tested (with an if statement) to check if it is equal to a certain value (week 4 covers **if/else** statements). You can print out the variable key (from within the **`void keyPressed() {}`** method) to see what characters relate to what keys… try it!!

Note: *some keys (like shift or enter) do not display recognizable characters, and must be detected using keycodes. For now we will not use these keys to do anything, so we can ignore this.*
https://processing.org/reference/key.html

Modify the **`void keyPressed() {}`** method in your program so that when you press each of the following keys, these messages will appear in the console:

pressing 'm' shows:

```
mirror mode is ON
```

pressing 'c' shows:

```
clearing draw area
```

pressing 'h' should re-print the showHelp() menu :

```
Drawing Instructions:
=========================
Click and hold mouse button down to draw in window
Release mouse button to stop drawing
press 'c' to clear the drawing
press 'm' to toggle mirrored mode
press 'h' to show this message
```

3. Now add the void **`mousePressed() {}`** and **`void mouseReleased() {}`** methods to your program so that the following messages are printed to the console when the mouse button is clicked and released (note, you don't need any if statements for this step because you don't need to check the value of any variables to know when the mouse is clicked).

   Clicking down on mouse button should print the following to the console:

   ```
   drawing mode is ON
   ```

   Releasing mouse button should print the following to the console:

   ```
   drawing mode is OFF
   ```

4. The **`mouseButton`** variable is a built-in variable (like `mouseX, mouseY, width, height`), that indicates which mouse button was actually clicked.
   https://processing.org/reference/mouseButton.html

First, modify your `mousePressed()` method to output a message that prints the value of `mouseButton`. If you have a mouse with multiple buttons, click them to print and see what number is associated with each. For e.g.:

```
drawing mode is ON
button = 37
drawing mode is OFF
drawing mode is ON
button = 39
drawing mode is OFF
```

Once you know which button is which, you will use the right mouse button (or any button on your mouse **other than** the one you will use to draw with). Use an if test on `mouseButton` to check if the right button was clicked, and if so, toggle the value of a boolean variable called **`standardPen`** between `true` and `false`. In other words…

If **`standardPen`** is currently `false`, and you click the right mouse button, it should change from `false` to `true` and output the following message:

```
standardPen is ON
```

if the **`standardPen`** is currently `true`, and you click the right mouse button, it should change from `true` to `false` and display the following message:

```
standardPen is OFF
```

Question 3: **[12 marks => 6 (min functionality) + 6 (challenge)]**

In this question, we will build a simple drawing app, that draws on the app window while the user is moving the mouse with the mouse button held down. First copy over your code from question 2, and modify as outlined below to achieve the required minimum functionality. After this is done, you may attempt to implement one (or more) of the suggested challenges. **You only need to do one of these challenges**, but you are free to try more than one of course.

MINIMUM FUNCTIONALITY (+ HINTS):

You can draw by detecting and setting a flag (e.g. a boolean variable called **`isDrawing`**), that is set to `true` when you click down on the mouse button, and switches to `false` when you release the mouse button (see question 2). Then you can check **`isDrawing`** in the **`draw()`** method, and if **`true`**, you can either draw a line from the previous mouse location (**`pmouseX`**, **`pmouseY`**) to the current mouse location (**`mouseX`**, **`mouseY`**), or draw a small filled circle/rect/ellipse at the current mouse location. Note, `pmouseX` and `pmouseY` are also built-in variables in processing.

When you are drawing with the left mouse button down (i.e. **`standardPen`** is **`true`**), you should draw in the colour BLACK (on a white background), whereas when you are drawing with the right mouse button down (when **`standardPen`** is **`false`**), you should draw in the colour RED.

Colours can be setup as constants for ease of use using the **`color()`** method, for example:

```
final int BLACK = color(0,0,0);
```

When you press the **`'c'`** key on the keyboard, this should clear the screen (return it to a white empty window).

CHALLENGES:

Explore and implement a **minimum of ONE** of the following challenges (you may do them all):

1. Create a rectangular area (within the app window) that you will restrict your drawing to – i.e. you will ONLY draw if the mouse button is clicked and held down within that draw area (this could be inside a smaller rectangle drawn within your app window). The method constrain() can help here.

2. Include additional key presses that can increase/decrease the size of the stroke/pen used to draw. Example if you are drawing with a line, change the stroke weight. If you are drawing with a circle/rect or ellipse, change the radius or length/width drawn)

3. Show a "status bar" in part of your application window (an area that displays text or some visual icon when you are drawing versus when you are not, and what the location of your cursor is in the draw area)

4. Mirrored drawing: when typing the 'm' key, you should draw two lines (one at the original mouse position, and one in a different colour that is mirrored in either x or y direction (i.e. the pen is mirrored (reflected) about the centre of the drawing area in either the horizontal axis or vertical axis at the centre of the draw area (see videos below for ideas)

    - basic function + mirrored drawing example [see video here]
    - status bar example [see video here]

*** if you add any key press options to your program, be sure to update your showHelp() method!*

STEP 3: SUBMISSION (DUE: Tuesday Oct 18, 2022 – 5:00pm)

NOTE: REMEMBER, you can choose to use the web-submit function (see Lab 0 for walkthrough). You will need to find and upload your lab3 *.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/