Lab Exam 3

There are 2 problems for you to solve. Please read *all the instructions carefully*.

We strongly advise you to do *everything exactly step-by-step*.
If you get stuck, you'll get partial credit for partial work -- so move on to another problem.
Please observe these restrictions, which our proctors will be monitoring:

- *No browsing the Internet (including the online p5.js reference).*
- *No browsing local files for code, documentation, or other material.*

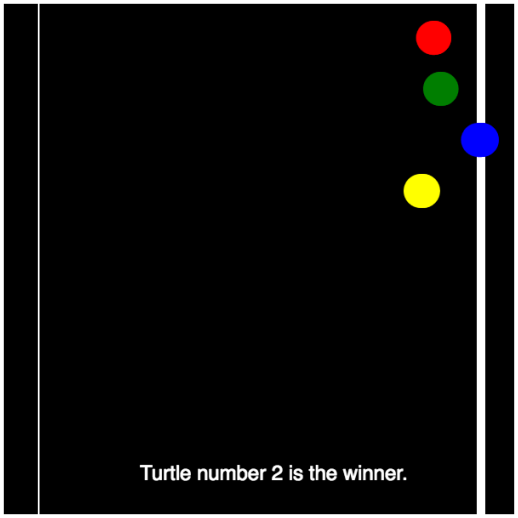
All work should be based on a provided starting template.
You can download a "starter" template, containing `part-a/sketch.js` and `part-b/sketch.js` from: **`http://bit.ly/2g7LFSL`**
Your solutions should be uploaded to Autolab as .zip files in the usual way.

Some Useful p5.js Functions

`createCanvas(w, h);`	*create a canvas with the given dimensions*
`width`	*the width of the canvas, once it has been created*
`height`	*the height of the canvas, once it has been created*
`key`	*the last key typed*
`background(r, g, b);`	*fill/erase the canvas with an RGB color*
`background(grayLevel);`	*fill/erase the canvas with a grayscale value*
`rect(x, y, w, h);`	*draw a rectangle*
`ellipse(x, y, w, h);`	*draw an ellipse or circle*
`line(x1, y1, x2, y2);`	*draw a line*
`point(x1, y1);`	*draw a point (size will be determined by `strokeWeight`)*
`rectMode(mode);`	*mode can be `CENTER`, `RADIUS`, `CORNER`, or `CORNERS`*
`ellipseMode(mode);`	*mode can be `CENTER`, `RADIUS`, `CORNER`, or `CORNERS`*
`colorMode(HSB);`	change fill(), stroke(), background(), and color() to use HSB system
`fill(r, g, b);`	*set the fill color for subsequent shapes to the color (r,g,b)*
`fill(grayLevel);`	*set the fill color for subsequent shapes to the specified graylevel*
`stroke(r, g, b);`	*set the color for subsequent lines or borders, to the color (r,g,b)*
`stroke(grayLevel);`	*set the color for subsequent lines or borders, to the graylevel*
`noFill();`	*choose that subsequent shapes will be drawn with no fill*
`noStroke();`	*choose that subsequent shapes will be drawn with no border*
`strokeWeight(pixels);`	*set the thickness for subsequent lines or borders*
`push();`	*save the current graphics transformation settings*
`pop();`	*restore the most-recently saved transformation settings*
`translate(x, y);`	*translate subsequent drawing by offsets x and y*
`scale(x, y);`	*scale by factors x and y (from the origin)*
`rotate(radians(degrees));`	*rotate by degrees (around the origin)*
`random(x);`	*get a random number between 0 and x*
`random(low, high);`	*get a random number in the range between low and high values*
`min(a, b);`	*get the lesser of the two numbers, a and b*
`max(a, b);`	*get the greater of the two numbers, a and b*
`map(val, inLo, inHi, outLo, outHi);`	*linearly re-maps a number, whose current range extends from inLo to inHi, to a new 'target' range, which extends from outLo to outHi.*
`function setup(){…}`	*a handler for what happens when the sketch first starts*
`function draw(){…}`	*a handler for what happens when the screen is refreshed*
`function keyPressed(){…}`	*a handler for what happens when the user presses a key*
`function mousePressed(){…}`	*a handler for what happens when the user clicks the mouse*
`noLoop();`	*disables continuous updating; draw() is executed only once.*
`mouseIsPressed, mouseX, mouseY`	*mouse input variables are updated before draw() is called*
operators:	*and:* `&&` *or:* `||`
	equality test: `==` or `===` *(for now, don't sweat the difference)*
	inequality test: `!=` or `!==`
	comparisons: `>` `<` `>=` `<=`

Problem A: Turtle Race

A. Modify the `part-a/sketch.js` file to implement a "Turtle Race" as shown in the following image:



B. Note that the turtle API is documented in the template sketch code. *Do not use methods `turnToward()` or `goto()`, whose documentation has been removed.*

C. (For an overview and grading criteria, see "E. Grading Criteria" below.) The following requirements should be met:
   a. Make the canvas 640 by 300 pixels.
   b. Draw a starting line where turtles start. (This is already implemented in the template sketch.)
   c. Draw a finish line where turtles finish. (This is already implemented in the template sketch.)
   d. Make an array of turtles to be in the race. (Recommendation: make a single turtle that completes the race before working with multiple turtles.)
   e. Each turtle is *automatically* created with a different color, so do not bother with colors.
   f. `makeTurtle` initializes turtles with the pen *down*, so be sure to put each turtle's pen *up* to avoid drawing trails when the turtles move forward.
   g. Your `draw()` function should draw each turtle by calling the *turtle*`.point()` method. (This *new* method draws a circle at *turtle*`.x`, *turtle*`.y`).
   h. The race starts when a key is pressed. The turtles sit on the starting line until the race starts. (This functionality is partially implemented; please see the function `keyPressed` in the template code.)
   i. While the race is on, each turtle advances by a random amount from 0 to 2, calculated by calling `random(2)`.
   j. The race ends when at least one turtle reaches the finish line. When that happens, call `raceFinish()`.

k. Implement `raceFinish()`. It should print on the canvas the winning turtle number, e.g. “Turtle number 0 won the race.” Then (still in `raceFinish()`) call `noLoop()` to stop drawing.

l. Two or more turtles may cross the finish line in the same frame (call to `draw()`). You do not have to consider this possibility, but if your program somehow picks one and only one winner, a small bonus will be awarded.

Hint i. Do not spend time on this unless you finish part B.

Hint ii. To generate a tie, temporarily modify your code so that each turtle moves exactly 20 on every frame. See if you get the expected result.

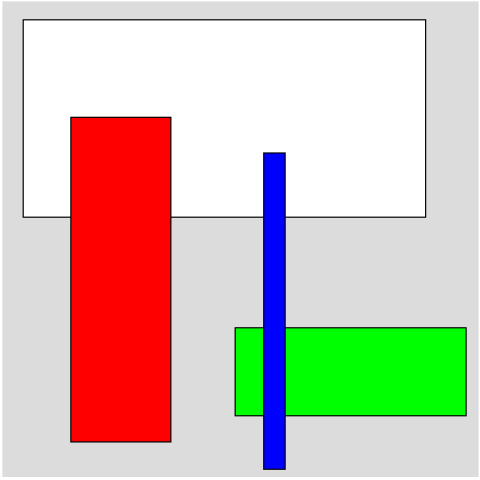
Hint iii. You may need an additional variable to remember the winner and a conditional to prevent reporting multiple winners.

D. **Grading Criteria**. There will be increasing points for each level of achievement. We recommend you complete level 1 or 2, submit your result, then work on higher levels in the time remaining:

1. A single turtle runs the race.
2. 4 racing turtles are stored in an array and run the race
3. 4 racing turtles are used and `for` loops are used to avoid writing the same commands 4 times.
4. `N` racing turtles are allowed, where `N` is a global variable set to some value. E.g. by setting `N` = 6, and making no other changes, the race will have 6 turtles. (At some point, turtles will run off the canvas or you will have to space them more closely – do not worry about this detail.)

Problem B: Drawing With the Mouse

A. Modify the `part-b/sketch.js` file to implement a (very) simple drawing program. The output might look something like this:



When the user clicks, the drawing program takes that location as the upper left corner of a rectangle. When the user clicks again, the location becomes the lower right corner of a rectangle. The rectangle is inserted at the end of an array. All of the rectangles in the array are displayed.

B. (For an overview and grading criteria, see “E. Grading Criteria” below.) The following requirements should be met (this is also an implementation guide; you can read and follow it step-by-step):
   a. Make the canvas size 400 by 400.
   b. Your program should in some way keep track of the number of mouse clicks.
   c. Your program should have an array to store rectangle objects, initially empty.
   d. Modify `draw()` to draw the rectangles objects in the array of rectangles.
   e. At this point, it would be smart to *manually* create a rectangle object, put it in the array, and debug your `draw()` function. Then comment out your test and proceed.
   f. If the number of mouse clicks becomes odd (hint: remember that `(x % 2)` is either 0 or 1), then remember the mouse coordinates in some way.
   g. If the number of mouse clicks becomes even, you now have 4 parameters for a rectangle: the mouse coordinates stored in part (b) above, and the current mouse X and Y coordinates. You can assume the current mouse location is below and to the right of the previously stored coordinates. Construct an object that stores these 4 parameters (they can be stored as x1, y1, x2, y2, or as x, y, w, h, using any names you desire.
   h. Take the object from part (g) above and insert it as the new last element of your array of rectangles. You already have `draw()`, so it will draw all the rectangles you have created.

C. EXTRA CREDIT
   a. Modify the rectangle object to store a color as well as coordinates.
   b. Modify the `draw()` command to fill the rectangle with its color.
   c. When the user types R, G, or B, set the "current color" to red, green, or blue. When a rectangle is created, set its color to the current color.
   d. Modify your code to display a rectangle "under construction," that is, start displaying the rectangle on the first click. The rectangle's lower right corner will track the mouse X and Y until the second click. Hint: you can create a rectangle on the first click and update it on every call to `draw()`, *or*, you can add additional draw code for the rectangle under construction and add it to the array only after the 2nd click. In this case, this special code will only be active when the click count is odd.
   e. Modify your "draw the rectangles" loop to use an expression of the form *the_rectangle*`.draw()`. Define a rectangle draw function and make sure that each rectangle object has a `draw:` attribute whose value is your new rectangle draw function, and make sure the new rectangle draw function uses "`this`" notation to refer to the rectangle object being drawn.

D. **Grading Criteria**: There are multiple levels of achievement, with increasing points, as follows:
   1. Draw a rectangle at coordinates determined by two mouse clicks.
   2. Save and draw each rectangle that is created by a pair of clicks.
   3. `draw()` must clear the canvas (with background()) and redraw all the rectangles each time it is called.
   4. EXTRA CREDIT: Rectangle color can be set by typing R, G, or B (for red, green, or blue) before the click for the upper left corner. (If you type R, G, or B again before the 2nd click, you can *optionally* change the color – it's up to you.)
   5. EXTRA CREDIT: All of the above, plus the rectangle is displayed and the lower right corner tracks the mouse until the 2nd click, which "freezes" the rectangle.
   6. EXTRA CREDIT: Modify your objects so that they contain a `draw` function as an *attribute*, in other words, to draw a rectangle in variable `rectangle`, you can write `rectangle.draw()`.