# CISC 1600, Lab 2.1: Processing

Prof Michael Mandel

## 1 Getting set up

For this lab, we will be using Sketchpad, a site for building processing sketches online using **`processing.js`**.

1.1. Go to http://cisc1600.sketchpad.cc in your browser

1.2. Create a new account and log in

### 1.1 Important information!

For this lab, you will be creating a different "sketch" for each section (except this first one). Each time you press the "Render" button, Sketchpad saves a version of your sketch and makes it accessible via a URL. Please make sure that you do this at least once after completing each step in the lab so that your work is saved properly. At the end of the lab, I will ask you to share the URLs of all of your sketches with me and I will go through the saved points.

## 2 Lines and coordinates

The first step is to figure out the coordinate system and learn how to draw lines going the directions you want them to.

2.1. Create a new sketch

- Click on the "New Sketch" button on the top right of the page
- Erase the existing code
- Replace it with this starting point

```
// Lab 2.1, Step 2.1: Create a new sketch
void setup() {
  size(300, 300);
  background(204);
}

void draw() {
  line( 10, 20, 30, 40 );
}
```

- Change the name of the sketch to "Lab 2.1, sketch 2" by clicking on the "[rename]" link at the top left of the page.

- Click on the "Render" (Play) button to see your work (and save it in the history)

2.2. Draw a horizontal line

- Update the call to `line()` so that the line is horizontal
- It should start at the same point and be of length 20
- Update the comment at the top of the sketch with information for the current step ("Step 2.2: Draw a horizontal line")
- Click on the "Render" button

2.3. Draw a vertical line

- Now update the call to `line()` so that the line is vertical
- It should start at the same point and be of length 20
- Update the comment at the top of the sketch and "Render" it

2.4. Draw one vertical line and one horizontal line meeting at the middle of the canvas and making a +

- The lines should extend from one side of the canvas to the other
- Use the `height` and `width` variables that Processing defines for you representing the height and width of the canvas.
- Update the comment at the top of the sketch and "Render" it

2.5. **Optional challenge:** Draw lines 10 pixels away from all four edges of the canvas

- Draw four lines using four calls to `line()`. Each line should be 10 pixels away from one edge of the canvas.
- You will again use `height` and `width`
- Update the comment at the top of the sketch and "Render" it

# 3 Using the mouse position

Now we will use the variables `mouseX` and `mouseY` that Processing automatically populates for us to track where the mouse is moving.

3.1. Create a new sketch. Change the name to "Lab 2.1, step 3". Replace the default code with the following:

```
// Lab 2.1, Step 3.1: Create a new sketch
void setup() {
```

```
  size(300, 300);
}

void draw() {
  background(204);
  line( 10, 20, 30, 40 );
}
```

3.2. Draw a line from 0,0 to the mouse

- Update the **`draw()`** function so that the line is drawn from the point (0,0) to the current position of the mouse. It should follow the mouse as it moves.
- The current position of the mouse is automatically stored in the variables **`mouseX`** and **`mouseY`** by Processing
- What happens if you comment out the call to **`background(204)`**?
- Update the comment at the top of the sketch and "Render" it

3.3. Draw a line from the middle of the canvas to the mouse

- Update the **`draw()`** function so that the line is drawn from the middle of the canvas to the current position of the mouse.
- The middle of the canvas can be found from the variables **`height`** and **`width`** that are automatically populated by Processing. Use them in a simple formula to find the middle.
- Update the comment at the top of the sketch and "Render" it

3.4. Draw a $20 \times 20$ square that has its top left corner at the current mouse position

- Update the **`draw()`** function to draw a square at the mouse's current coordinates. Use the **`rect()`** function to do so. The function takes four numerical arguments: **`x, y, width, height`**.
- Update the comment at the top of the sketch and "Render" it

3.5. **Optional challenge:** Draw a square with its center at the mouse's position. Hint: call **`rectMode(CENTER)`** in your **`setup()`** function.

3.6. **Optional challenge:** Draw a rectangle centered on the middle of the canvas with one corner at the mouse's position. Hint: using **`rectMode(CENTER)`**, you can plot the rectangle at a constant position (**`x, y`**), but with height and width that depend on the mouse position.

# 4 Changing colors

Now we will use the mouse position variables to explore the use of color in Processing.

4.1. Create a new sketch. Change the name to "Lab 2.1, step 4". Replace the default code with the following:

```
// Lab 2.1, Step 4.1: Create a new sketch
void setup() {
  size(300, 300);
  colorMode(RGB, width);
  background(204);
}

void draw() {
}
```

4.2. Change background gray-level based on mouse position

- Add a call to **`background()`** in the **`draw()`** function, pass it a single argument, the current mouse x-coordinate
- What happens when you move the mouse around the canvas? What happens when you move it up and down? Side to side?
- Update the comment at the top of the sketch and "Render" it

4.3. Change gray-level of a shape based on position

- Change the call to **`background()`** that you just added to have a fixed argument of 204 again
- Update the **`draw()`** function to draw a circle 50 pixels in diameter centered at the current mouse location. Use the **`ellipse()`** function, which takes four arguments: **`x, y, width, height`**.
- Add a call to **`fill(mouseX)`** to **`draw()`** between drawing the background and drawing the circle
- What happens when you move the mouse around the canvas? What happens when you move it up and down? Side to side?
- Update the comment at the top of the sketch and "Render" it

4.4. Change color based on position

- Replace the call to **`fill()`** with **`fill(mouseX, mouseY, 0)`**
- What happens when you move the mouse to the four corners of the canvas? Can you explain that based on the ideas of color mixing from lecture?

- Comment out the call to **`background(204)`** and move the mouse around the canvas again.
- Update the comment at the top of the sketch and "Render" it

4.5. **Optional challenge:** Change transparency of the circle based on the mouse position. Hint: the transparency is set by an additional last argument to **`fill()`** that ranges between 0 (opaque) and 255 (totally transparent).

# 5 The `if` statement

5.1. Create a new sketch. Change the name to "Lab 2.1, step 5". Replace the default code with the following:

```
// Lab 2.1, Step 5.1: Create a new sketch
void setup() {
  size(300, 300);
  noStroke();
  fill(255, 0, 0);
}

void draw() {
  background(204);
  rect(150, 0, 150, 300);
}
```

5.2. Draw on the half of the canvas that the mouse is in

- You will need to replace the call to **`rect()`** with an **`if`** statement:

```
if (condition) {
  // statements to execute if condition is true
} else {
  // statements to execute if condition is false
}
```

- Figure out what **`condition`** should be so that it is **`true`** when the mouse is in the left half of the canvas and **`false`** when it is not. Add it to the **`if`** statement.
- In the **`if`** clause, write the code to draw a rectangle covering the left half of the canvas.
- In the **`else`** clause, draw a rectangle covering the right half of the canvas
- Render the sketch and move the mouse around the canvas. If the rectangle flips sides based on the position of the mouse, you have

implemented this correctly.

- Update the comment at the top of the sketch and "Render" it

5.3. Draw on the quarter of the canvas that the mouse is in

- Figure out what arguments you need to pass to **`rect()`** to draw the red rectangle on each quarter of the canvas (top left, top right, bottom left, bottom right)
- Figure out what tests on the mouse coordinates determine whether the mouse is in each quarter of the canvas. Hint: it will be a combination of a test on **`mouseX`** AND a test on **`mouseY`**. Use the **`&&`** operator to implement AND.
- Expand your **`if`** statement with two **`else if`** clauses. Update the test conditions in the **`if`** and **`else if`** statements to reflect the test for whether the mouse is in each quarter of the canvas.
- Insert the appropriate call to **`rect()`** inside each clause so that the rectangle is drawn on the quarter of the canvas where the mouse pointer is currently
- Update the comment at the top of the sketch and "Render" it

5.4. **Optional challenge:** Highlight a box if the mouse is inside of it.

- Draw a rectangle in the middle of the canvas
- Figure out what test to perform to see if the mouse is inside of the square. Hint: test if the mouse is on the desired side of each of the four edges of the box and AND them together using the **`&&`** operator
- Call the function **`fill()`** with a light color if the test returns **`true`** and a dark color otherwise.
- Update the comment at the top of the sketch and "Render" it

# 6 Looping statements

6.1. Create a new sketch. Change the name to "Lab 2.1, step 6". Replace the default code with the following:

```
// Lab 2.1, Step 6.1: Create a new sketch
void setup() {
  size(500, 500);
}

void draw() {
```

```
  background(204);
}
```

6.2. Draw five boxes stacked vertically using five calls to **`rect()`**

- Each box should be $40 \times 40$ pixels.
- Adjacent boxes should be separated by 20 pixels
- Update the comment at the top of the sketch and "Render" it

6.3. Draw the same five boxes stacked in the same way using a **`for`** loop.

- Figure out a formula for the position of each box as a function of the box "number" (first = 0, second = 1, third = 2, ...)
- Write a **`for`** loop utilizing your formula. The **`draw()`** function should look something like this:

```
void draw() {
  background(204);
  for (int i = 0; i < 5; i++) {
    // Your code here to draw box number i
  }
}
```

- What happens if you start the loop at **`i = 1`** instead of **`i = 0`**? What if the loop uses the test **`i <= 5`** instead of **`i < 5`**?
- Update the comment at the top of the sketch and "Render" it

6.4. Draw the same five boxes stacked in the same way using a **`while`** loop.

- Update your **`draw()`** function to look like this

```
void draw() {
  background(204);
  int i = 0;
  while (i < 5) {
    // Same code as step 6.3 here to draw box number i

    i = i + 1;
  }
}
```

- Update the comment at the top of the sketch and "Render" it

6.5. **Optional challenge:** Modify the **`while`** loop example to draw boxes until they go off the edge of the canvas. Hint: you will need to change the condition that the **`while`** loop tests.

# 7 Functions: reusing code

7.1. Create a new sketch. Change the name to "Lab 2.1, step 7". Replace the default code with the following:

```
// Lab 2.1, Step 7.1: Create a new sketch
void setup() {
  size(300, 300);
}

void draw() {
  background(204);
}
```

7.2. Draw a house

- Update your **`draw()`** function to draw a house using calls to **`rect()`** and **`triangle()`** (which takes six arguments, the position of the three vertices: $x_1, y_1, x_2, y_2, x_3, y_3$)
- Update the comment at the top of the sketch and "Render" it

7.3. Make your code into a function to draw a house

- Use this function skeleton

  ```
  void house() {
    // Insert your code here to draw a house
  }
  ```

- Take your code from **`draw()`** that draws the house (not the background or anything else) and move it to the **`house()`** function.
- Insert a call to the **`house()`** function in **`draw()`** where that code used to be.
- Update the comment at the top of the sketch and "Render" it

7.4. **Optional challenge:** Draw the house relative to a particular "center" position

- Update the definition of the house function to look like this

  ```
  void house(int x, int y) {
    // ...
  ```

- Update the call to **`house()`** in **`draw()`** to pass in the **`x`** and **`y`** arguments. The call to house should now look like this

  ```
  void draw() {
    // ...
  ```

```
  house(0, 0);
}
```

- Add `x` to all of the x coordinates in the calls to `rect()` and `triangle()` in the function. Similarly add `y` to all of the y coordinates. What does this do?
- In the `draw()` function, change the call to your function to `house(50, 20)`. What happens?
- Update the comment at the top of the sketch and "Render" it

7.5. **Optional challenge:** Make the house follow the mouse

7.6. **Optional challenge:** Use your function to draw several houses at fixed positions. Hint: you will need to call `house()` several times with different arguments in your `draw()` function.

# 8 Create a project in Thimble with links to your sketches

8.1. Create a new Thimble project

- Give it a reasonable title and `<h1>`, remove the default paragraph.
- Add an unordered (`<ul>`) or ordered (`<ol>`) list in the HTML page

8.2. Find links from each of your Sketchpad sketches

- For each sketch, click on the yellow "Share" button
- In the "Share" section of the sidebar, find the "link to this revision" link. Copy the url and paste it into your thimble project.
- Find the blue timeline with squares labeled "p" on it and use the slider on that timeline to find the points at which you had completed each sub-part of each part of the lab. Use your updated comments at the top of the sketch to figure out when this was.
- Copy the "link to this revision" link for each step's final version of the sketch and paste the URL into your thimble project

8.3. Organize your links in Thimble

- In your Thimble project, create a list item (`<li>`) for each sketch.
- Create a link (`<a href="...">...</a>`) to the final version of that sketch in that list item.
- Create a sub-list (using `<ul>` or `<ol>`) for each item by creating a nested unordered or ordered list inside of the existing one

- Create links to each of the sub-steps of each sketch in those sub-items.

# 9 Optional challenge: Extensions

As an extra extra challenge, implement **two** or more of the following suggested sketches.

- Draw 5 squares evenly spaced on the line between the point (0,0) and the current mouse position.
- Draw a checkerboard pattern. Hint: use two "nested" loops (one inside the other).
- Draw a checkerboard pattern out of equilateral triangles. Hint: alternating rows will need to be offset from each other by half of the length of a triangle side.
- Draw several shapes that all move in different directions and different speeds based on the position of the mouse. For example, one might move up when the mouse moves up and another might move down.
- Draw something cool of your own choosing.
- Draw something cool of your own choosing that changes as the mouse moves.