CISC 1600, Lab 2.2: Interactivity in Processing

Prof Michael Mandel

1 Getting set up

For this lab, we will again 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. Log in to the account that you created for Lab 2.1.

1.1 Important information (reminder)!

For this lab, you will be creating a different "sketch" for each section (except this first one). Each time you press the "Render" button (the button with the "play" symbol on it), 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.

1.2 Create a project in Thimble with links to your sketches

Create a new project in Thimble for this lab. In the HTML file, there should be a list for each sketch and a nested sub-list containing the individual steps for that sketch. After completing and rendering each step, copy the link to that step into your thimble project. For details, see the previous lab.

2 Mouse variables

In this sketch, we will use the mouse variables that Processing automatically populates for us. We have already seen **`mouseX`** and **`mouseY`** in the last lab, this step we will use **`mousePressed`** and **`mouseButton`**.

2.1. Create a new sketch. Change the name to "Lab 2.2, step 2". Replace the default code with the following:

```
// Lab 2.2, Step 2.1: Create a new sketch
void setup() {
  size(300, 300);
}
```

```
void draw() {
  background(204);
  rect(width/4, height/4, width/2, height/2);
}
```

2.2. Change the color of the square if the mouse is clicked

- Insert an **`if`** statement in the **`draw()`** function before the call to **`rect()`**, which should remain outside of the **`if`** statement's braces ({})
- The **`if`** statement's condition should test whether the variable **`mousePressed`** has the value of **`true`**
- If that condition is met, the current fill color should be set to gray level 230 using the **`fill()`** function
- Otherwise (**`else`**), the fill color should be set to gray level 255 using the **`fill()`** function
- Update the comment at the top of the sketch and "Render" it

2.3. Change the color of the square only if the mouse is clicked inside of it

- We will modify the code from the previous step to only change the color of the box if the mouse is clicked inside of it
- There are four conditions that must be met when the mouse is inside of the box, one for each edge of the box. Figure out what they are.
- In order to figure out whether all four conditions are met at the same time, AND them all together using three instances of the **`&&`** operator
- Using a fourth **`&&`** operator, combine these four conditions with the original condition in the **`if`** statement that tests if the mouse button is pressed
- Try clicking inside the box and above, below, left, and right of it. Make sure that the box only changes color when you click inside of it.
- Update the comment at the top of the sketch and "Render" it

2.4. **Optional challenge:** Change the color of the square depending on which mouse button is clicked

- Now we are going to add an additional **`if`** statement inside of your existing **`if`** statement to change the fill color of the box based on which mouse button is pressed
- When the mouse is clicked, Processing automatically sets the value of the variable **`mouseButton`** to one of these three predefined values representing mouse buttons: **`LEFT`**, **`RIGHT`**, and **`CENTER`**. To test which mouse button is clicked, use a test such as **`mouseButton == LEFT`**

- Add this new **if** statement so that it is run if the mouse is clicked inside of the box
- Write the **if** statement so that if the left mouse button is clicked, the code sets the fill color of the rectangle to 230 using the **fill()** function
- Otherwise (**else**), set the fill color to 170 using the **fill()** function
- Try clicking different mouse buttons and make sure that the color changes appropriately
- Update the comment at the top of the sketch and “Render” it

3 Keyboard variables

Processing also populates several keyboard variables for you, depending on what the user is typing: **keyPressed**, **key**, and **keyCode**. We will also use the **text()** function to write on the canvas.

3.1. Create a new sketch. Change the name to “Lab 2.2, step 3”. Replace the default code with the following:

```
// Lab 2.2, Step 3.1: Create a new sketch
void setup() {
  size(300, 300);
  textSize(60);
  textAlign(CENTER);
}

void draw() {
  background(204);
  text("None", width/2, height/2);
}
```

3.2. Write a message to the canvas when a key is pressed

- Write an **if** statement in **draw()** after the call to **background()**. The condition should test whether the variable **keyPressed** is **true**. To test for equality, use the **==** operator.
- Move the existing call to **text()** inside the **else** clause of the **if** statement, so that it is only called if a key is NOT pressed.
- If a key IS pressed, add a new call to **text()** that displays the string ”Pressed”
- Try out the sketch. Make sure that it says “Pressed” when you type letter keys. What does it say when you press just the Shift key?

Control? Backspace? Enter? Up? Down?

- Update the comment at the top of the sketch and “Render” it

3.3. Show which key is pressed

- Update your code so that instead of always printing the same message, it prints the key that was pressed.
- Use the variable **`key`** in the call to **`text()`** when the **`if`** condition is true
- Try out the sketch. Make sure that it prints the right character when you type letter keys. What does it say when you press just the Shift key? Control? Backspace? Enter?
- Update the comment at the top of the sketch and “Render” it

3.4. Only show printable keys

- Some keys on the keyboard don’t correspond to characters that we would want to print, for example, Shift, Backspace, Up, Down. At the moment, when one of those keys is pressed, no character is printed (actually a blank character is printed)
- Update the test in your **`if`** statement so that it tests that a key is pressed AND that the variable **`key`** is not equal to the value **`CODED`**. If it is equal to **`CODED`**, then the key is not printable. To test for inequality, use the **`!=`** operator.
- Try out the sketch. Make sure that it prints the right character when you hold letter keys and that it prints “None” when you press Shift, Control, Up, and Right, but not for Backspace, and Enter (they are printable). What happens when you hold Shift and press the “1” key to type an exclamation point?
- Update the comment at the top of the sketch and “Render” it

3.5. **Optional challenge:** Show the last key that was pressed

- Now we will define our own variable to save the value of the last key that was pressed
- Insert the following line BEFORE the **`setup()`** function:

  ```
  char lastKey = ' ';
  ```

  This declares a new variable called **`lastKey`** that holds a single character. It also stores the space character in the variable to begin with. We will read and write it inside the **`draw()`** function, but by declaring it outside of any functions, it becomes a global variable, meaning that its value will be saved between calls to **`draw()`**

- Now when the **if** condition is true, instead of calling **text()** directly, save the value of **key** in the new **lastKey** variable that you just created
- Remove the **else** clause entirely
- Write a new call to **text()** after the end of the **if** statement drawing the character **lastKey**
- Try out the sketch. Make sure that it prints the right character when you type letter keys, and that they remain on the screen when you let go of the key.
- Update the comment at the top of the sketch and "Render" it

4 Mouse events

In this sketch, we will define new functions to handle various mouse events: **mousePressed()**, **mouseReleased()**, **mouseMoved()**, and **mouseDragged()**. If we create a function with one of these standard names, it will be called when a relevant mouse event occurs.

4.1. Create a new sketch. Change the name to "Lab 2.2, step 4". Replace the default code with the following:

```
// Lab 2.2, Step 4.1: Create a new sketch
int gray = 0;

void setup() {
  size(300, 300);
}

void draw() {
  background(gray);
}

void mousePressed() {
}
void mouseReleased() {
}
```

4.2. Lighten the background color each time the mouse is pressed

- Inside the **mousePressed()** function, set the value of the **gray** variable to be its current value plus 20.
- Test the sketch. What happens when you click the mouse on the canvas? What happens when you click and hold?

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

4.3. Lighten the background color each time the mouse is released

- Delete the **`mousePressed()`** function you just created
- Inside the **`mouseReleased()`** function, set the value of the **`gray`** variable to be its current value plus 20.
- Test the sketch. What happens when you click the mouse on the canvas? What happens when you click and hold?
- Update the comment at the top of the sketch and "Render" it

4.4. Draw a square at each place the mouse is clicked

- Update the sketch to have the following contents only

```
// Lab 2.2, Step 4.4: Draw a square at each place the mouse is clicked
void setup() {
  size(300, 300);
  fill(0, 102);
}

void draw() {
  // Empty draw() keeps the program running
}

void mousePressed() {
}
```

- Inside the **`mousePressed()`** function, call **`rect()`** to draw a 30×30 square at the current mouse coordinates (remember the variables **`mouseX`** and **`mouseY`** from the previous lab).
- Test the sketch. What happens when you click on the canvas? What happens when you click close to an existing square? Why is that?
- Update the comment at the top of the sketch and "Render" it

4.5. **Optional challenge:** Draw a square on the canvas that the user can drag to a new position.

- Clear your existing code and create appropriate **`setup()`**, **`draw()`** and **`mouseDragged()`** functions.
- Create two global variables to store the current coordinates of the square
- Update the coordinates inside the **`mouseDragged()`** function, but only if the click is inside of the square

- Remember that the user probably won't click right on the point specified by the coordinates that you store for the square. You'll need to figure out where in the square they have clicked and compensate for the offset from that point before updating the coordinates.
- Update the comment at the top of the sketch and "Render" it

5 Keyboard events

The keyboard events **`keyPressed()`** and **`keyReleased()`** act similarly to the corresponding mouse events. Let's use them to make a little interactive "game".

5.1. Create a new sketch. Change the name to "Lab 2.2, step 5". Replace the default code with the following:

```
// Lab 2.2, Step 5.1: Create a new sketch
int squareSize = 30;
int squareX, squareY;

void setup() {
  size(300, 300);
  rectMode(CENTER);
  squareX = width/2;
  squareY = height/2;
}

void draw() {
  background(204);
  rect(squareX, squareY, squareSize, squareSize);
}

void keyPressed() {
}
```

5.2. Make a square grow each time you press the enter key

- Add an **`if`** statement inside the **`keyPressed()`** function. Its condition should be that the variable **`key`** is equal to the value **`ENTER`**
- If the condition is met, increase the value of the variable **`squareSize`** by 10
- Test the sketch. The square should grow when you press the "Enter" key, but not any other key
- Update the comment at the top of the sketch and "Render" it

5.3. Move a square around the screen using the arrow keys

- Now insert a second `if` statement after the end of the first `if` statement. Its condition should be that the variable `key` is equal to the value `CODED` and that the variable `keyCode` is equal to the value `UP`. In this case, you should move the square up by changing the value of either `squareX` or `squareY` by 10. Figure out which one to change and whether to increase it or decrease it.
- Add an `else if` clause that tests whether `key` is `CODED` and `keyCode` is `DOWN`. Figure out which variable to change and which direction to change it to move the square down.
- Add similar `else if` clauses for `LEFT` and `RIGHT`
- Update the comment at the top of the sketch and "Render" it

5.4. **Optional challenge:** Draw a different shape based on the last key pressed

- Add a variable called `lastKey` of type `char`
- In `keyPressed()` add code at the end so that if `key` is not equal to `CODED`, save it to `lastKey`
- Update the `draw()` function to draw a different shape depending on the value of `lastKey`. If it is 'c' draw a circle, 'e' draw an ellipse, 'l' draw a line, 'p' draw a point, 't' draw a triangle. Otherwise, draw a square.
- Update the comment at the top of the sketch and "Render" it

6 Submit it

Publish your Thimble project with links to all of your sketches and submit the URL for the published project to the Blackboard dropbox.