# Lab Week 3. Events and Animation

### Last Week

* Modular code
  + Procedures
  + Functions (see videos and walkthrough in week 2)
  + Passing parameters
* Local variables, global variables

## Learning Objectives

* Events
* Animation
* IF statement – conditional branching
* User interaction
* Pong Game

You should have completed all last week’s practical lab exercises. How are you progressing with the formative tests?

[Note : you may need to click on the drawn screen to interact with it – via mouse or keyboard]

## Resources

* Lecture Notes & moodle – some references on conditionals (**if** statements) on moodle
* Processing website - reference
* <http://www.cs.sfu.ca/CourseCentral/166/tjd/first_program.html>

[Pay, particular attention to the exercises at the bottom of the page above – which you should attempt in your own time – some of these exercises come from that source]

Save your code after each exercise

//Exercise 1.

void setup()

{

size(500, 500);

}

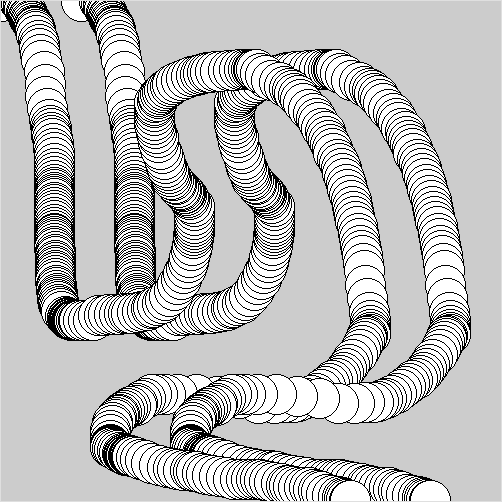
void draw()

{

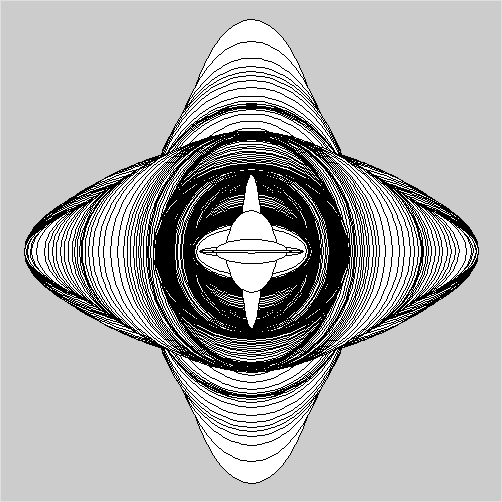
ellipse(mouseX, mouseY, 40, 40);

}

Enter the code precisely as above, run and move your mouse over the screen.



**Ex2** Alter the code above so that we see two parallel lines of circles, one offset, to the right, from the first by 50 pixels as in the picture.

Ex3. Alter the code so that ellipse remains in the centre of the screen, but the mouseX and mouseY alter the width and height of the ellipse.

Add an **if** statement, so the width and height have a max value of 100.

**Ex4.** Write a program which draws 3 concentric circles at the mouse position (leaving no trail). Remember to consider where the background command is placed, this commands clears the screen in a given colour.

**Ex 5.** Write a program which allows the UP and DOWN keys to move a rectangle (**rect** command) up and down the screen (draw it near the left hand edge). We covered this event in the lecture. Add **if** statements to prevent it disappearing off the top and bottom edge (try to keep the rectangle completely within the screen).

What do we need to store to know where to draw the rectangle?

**Ex6. Bouncing ball**. In the lecture we started thinking about making a ball bounce off the left and right edge of the screen. We now want to extend this so the ball moves up and down (with bounces from top and bottom) as well.

Think about the extra variables we will need. Hint everything we tackled for the **x** movement needs duplicating for the **y** movement.

//Global variables

int x; //ball x position

int deltaX = 5; //ball x direction is right, step 5

void **setup**() //runs once at start

{

**size**(500,250);

}

void **draw**() //runs repeatedly

{

**background**(200); //clear screen RGB = 200 (grey)

//draw and update ball position

**ellipse**(x,125,10,10);

x = x + deltaX; //move ball x right

//Collision Detection

//collide right hand edge?

if (x>=500)

deltaX = -deltaX; //reverse x direction

//collide left hand edge?

//collide top edge?

//collide bottom edge?

}

Think about refactor code above, moving code to a separate procedure **updateBall()**, **bounceOffWalls()** as another.

**Ex 7.** Modularise code and add a Bat

In Ex 5, we moved a rectangle up and down using the arrow keys. Add this code to your ball program and add in collision detection for the bat. Disable bouncing from the left hand edge and you have a Pong style squash game.

**Extension exercises**

Using the Processing web pages for reference you can extend this to a complete game by keeping a score (displayed on screen, **text** command) of the number of shots in a rally, add in lives, 3 misses and game over etc.

* Extend your game to make it two player.
* Add a second ball to the game.
* Once completed try to write a simple algorithm to allow computer control of one of the players (simple Artificial Intelligence – how can we achieve this with the techniques we have met so far).

**EtchySketch**

The code below allows the user to draw a continuous picture on the screen, using the arrow keys. This code can be adapted to become a **light cycles game**. We would need to test the colour of the new position before the point is drawn there. The get() function (see <https://processing.org/reference/get_.html>) returns the colour of a pixel on the screen, allowing you to check whether the new position is already occupied or a background colour.

int x,y; //position

int speedX=0; //ready for liteCycles

int speedY=0;

void setup()

{

size(400,400);

x=width/2;

y=height/2;

}

void draw()

{

point(x,y);

}

void keyPressed()

{

if(key==CODED) //if arrow cluster pressed

{

if (keyCode==UP)

y=y-1;

if (keyCode==DOWN)

y=y+1;

if(keyCode==LEFT)

x=x-1;

if(keyCode==RIGHT)

x=x+1;

}

}

CodingBat – function writing practice

<https://codingbat.com/java/Logic-1>

pick up where you left off with the problems in the link above, there are also more complex problems here:

<https://codingbat.com/java/>