

LCC 6310
The Computer as an
Expressive Medium

Lecture 3

Suggestions on learning to program

Spend a lot of time fiddling around with code

Programming is something you have to learn by trying it

Get help from other people

I expect those who already know some programming to help others

Figure things out in groups

Ask me questions in class

Ask Micah questions in tutorial

Overview

Programming concepts

Built-in Processing methods you fill in

Loops

Reading the time

Arrays

Questions about assignment 1

So far...

We've seen how to write commands in Processing

We've learned that commands need to end with a ;

We've learned how to call **methods** with specific **arguments**

We've learned how to **declare variables** and **assign values** to them

We've learned how to **print values** to the text output area

We've learned about the **primitive variable types**

We've learned how to control flow with conditional **if-then** statements

By default, the sequence of commands we write in Processing will execute once and then the program will terminate

This is Processing's "Basic" mode

Basic mode example

```
size(200, 200);  
background(255);  
  
color yellowish = color(255, 204, 0);  
int x = 30;  
int y = 20;  
int squareSide = 50;  
  
noStroke();  
fill(yellowish);  
rect(x, y, squareSide, squareSide);
```

Let's try it out in [Processing...](#)

Continuous mode

Processing's continuous mode provides:

A `setup()` structure that is run once when the program begins, and

A `draw()` structure that continually loops through the code inside

This enables writing custom methods and classes and using keyboard and mouse events... more on this later!

First, some more about `setup()` and `draw()`...

setup()

`setup()` is called once when a sketch first starts executing

Place any startup code in `setup()`, e.g.

- Setting the size
- Setting the background color
- Initializing variables
- ...

draw()

`draw()` is called continuously by the program

Put code in `draw()` when you need to constantly update the display (for example, animating an object)

Example of setup() and draw()

```
int x, y; // declare integer variables x and y

void setup() { // QUESTION: why do we need the void?
  size(400, 400); // set the window size to 400x400 pixels
  background(0); // set background to black
  x = 0;
  y = height/2; // height is a system variable, see reference
}

void draw() {
  background(0);
  ellipse(x, y, 20, 20);
  x = x + 1;
  if (x > width) { // width is a system variable, see reference
    x = 0;
  }
}

QUIZ: What does this program do?
Let's try it out in Processing!
```

Controlling draw()

frameRate() can be used to set the number of times per second that draw() is called

frameRate(30) says to call draw() 30 times a second (if the computer is capable of it)

delay() delays execution for a certain number of milliseconds

delay(250) delays for 250 milliseconds (1/4 of a sec.)

You can use delay() or frameRate() to determine how fast you want draw() to be called – frameRate() is probably easier

noLoop() tells the system to stop calling draw()

If you want to, for example, turn off animation

loop() tells the system to start calling draw() again

Use noLoop() and loop() together to turn drawing on and off

framerate()

```
void setup() {
  frameRate(15); // try out different fps values!
}

int pos = 0; // QUICK QUIZ:
// what happens if you declare pos inside setup()?

void draw() {
  background(255);
  pos = pos + 1;
  line(pos, 0, pos, 100);
  if (pos > width) {
    pos = 0;
  }
}
```

delay()

```
int pos = 0;

void draw() {
  background(255);
  pos = pos + 1;
  line(pos, 0, pos, 100);
  if (pos > width) {
    pos = 0;
  }
  delay(250); // stops the program for 250 milliseconds
              // try out different delay values
}
```

noLoop()

```
boolean looping = true; // try changing this to false!
int i = 0;

void setup() {
  if (!looping) {
    noLoop();
  }
}

void draw() {
  println("count "+i);
  i = i + 1;
}
```

Getting info from the mouse

mouseX and mouseY – variables that automatically contain the current mouse location

pmouseX and pmouseY hold the previous location

mousePressed – boolean variable that is true if the mouse button is down

mouseButton – value is LEFT, RIGHT or CENTER depending on which button is held down

Let's look at some examples...

mouseX and pmouseX

Horizontal motion

```
void draw() {
  background(204);
  line(mouseX, 20, mouseX, 80);
}
```

What happens if you make this change?

```
void draw() {
  background(204);
  line(mouseX, 20, pmouseX, 80);
}
```

mousePressed and mouseButton

```
void draw() {
  if (mousePressed && (mouseButton == LEFT)) {
    fill(0);
  } else if (mousePressed && (mouseButton == RIGHT)) {
    fill(255);
  } else {
    fill(126);
  }
  rect(25, 25, 50, 50);
}
```

Mouse methods

There are several built-in methods you can fill in to process mouse events

mousePressed(), mouseReleased(), mouseMoved(), mouseDragged()

Let's look at an example...

mouseMoved()

```
int value = 0;
boolean goingUp = true;

void draw() {
    fill(value);
    rect(25, 25, 50, 50);
}

void mouseMoved() {
    value = (goingUp) ? (value + 5) : (value - 5); // what the !?!?!
    if (value > 255) {
        goingUp = false;
    } else if (value <= 0) {
        goingUp = true;
    }
}
```

Loops

Sometimes you want to execute code multiple times

E.g. draw() is being called in a loop

Java provides a number of looping mechanisms

They all test some boolean expression (just like an if statement does) and continue to execute code while the expression is true

while loops

```
while(<boolean exp>) {
    <code to execute multiple times>
}
```

Executes the code within the curly brackets while the boolean expression is true

for loops

```
for(<init statement>; <boolean exp>; <final statement>) {  
    <code to execute in loop>  
}
```

First executes the initialization statement

Then tests the boolean expression – if it's true, executes the code inside the curly brackets once

Then repeats the following: execute final statement, test boolean expression, execute code if true

This structure is often used to execute a block of code a specific number of times as follows:

Init statement -> start counter, e.g. `int counter = 0;`

Boolean exp -> looping condition, e.g. `counter < 10;`

Final statement -> increment counter, e.g. `counter++;` (same as `counter=counter+1;`)

Converting for to while

Seeing how *for* loops can be converted to *while* loops helps you understand *for*

```
for(<init statement>; <boolean exp>; <final statement>) {  
    <code>  
}
```

is the same as

```
<init statement>  
while(<boolean exp>) {  
    <code>  
    <final statement>  
}
```

Reading time

`int hour()` – returns the hour (0 – 23)

`int minute()` – returns the minutes (0 – 59)

`int second()` – returns the seconds (0 – 59)

`int day()` – returns the day of the month (1 -31)

`int month()` – returns the month (1 – 12)

`int year()` – returns the four digit year (e.g. 2004)

`long millis()` – returns number of millis since start of app

Let's try some of these in [Processing...](#)

draw() has nothing to do with time

The value returned by `second` (or `milliseconds()`) has nothing to do with how often `draw()` is called

In `draw()` you draw frames – you don't know how often it will be called

Put a `println` in the loop to see how often it gets called

```
long lastTimeLoopWasCalled = 0;  
void draw() {  
    long milliseconds = millis();  
    println(milliseconds - lastTimeLoopWasCalled);  
    lastTimeLoopWasCalled = milliseconds;  
}
```

Arrays

Hold a series of data elements of the same type

One of the simplest data structures, think of it as a convenient way to store a whole lot of elements of the same type

You can then access any particular element if you know at which position in the array it is located, i.e. its *index*

Effect of creating an int variable

Code

```
// Single int
int anInt;
```

Effect

Name: `anInt`, Type: `int`



```
// Put a value in the int
anInt = 3;
```

Name: `anInt`, Type: `int`



```
// Type error!
anInt = "hello";
```

Name: `anInt`, Type: `int`



Can't shove "hello" into an int

Effect of creating an array of ints

Code

```
// declare int array
int[] intArray;

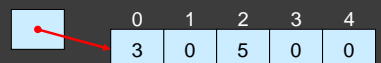
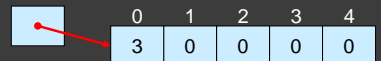
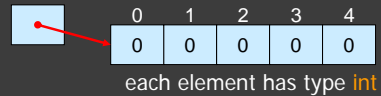
// initialize int array
intArray = new int[5];

// set first element
intArray[0] = 3;

// set third element
intArray[2] = 5;
```

Effect

Name: `intArray`, Type: `int[]`



Practice reading code

If code is a medium, then it can be both written and read

Reading code reveals

- New programming constructs
- Strategies and techniques (design patterns)
- Style
- Philosophical assumptions (deep reading)

Let's figure out the Motion -> [BrownianMotion](#) example together

- Erratic or random motion of small particles in a fluid
- The mathematical models used to describe these movements

```

int num = 2000;
int range = 4;

float[] ax = new float[num];
float[] ay = new float[num];

void setup() {
  size(200, 200);
  for(int i=0; i<num; i++) {
    ax[i] = 50;
    ay[i] = height/2;
  }
  frameRate(30);
}

void draw() {
  background(51);


  // Shift all elements 1 place to the left
  for(int i=1; i<num; i++) {
    ax[i-1] = ax[i];
    ay[i-1] = ay[i];
  }

  // Put a new value at the end of the array
  ax[num-1] += random(-range, range);
  ay[num-1] += random(-range, range);

  // Constrain all points to the screen
  ax[num-1] = constrain(ax[num-1], 0, width);
  ay[num-1] = constrain(ay[num-1], 0, height);

  // Draw a line connecting the points
  for(int i=1; i<num; i++) {
    float val = float(i)/num * 204.0 + 51;
    stroke(val);
    line(ax[i-1], ay[i-1], ax[i], ay[i]);
  }
}

```



New constructs

Array declaration and reference

```

float[] a = new float[10];
for(int i=0; i<10; i++) {
  a[i] = 3;
}
print(a[3]);

```

Math operations and functions

```

float val = (float)10/3 * 2.0 + random(0,10);
print(val);

```

Type casting

```

float val = (float)10/3;
print(val);

```

Assignment 1

Posted [online](#), due Friday

- A1-01: Draw three lines.
- A1-02: Draw five lines.
- A1-03: Draw three ellipses.
- A1-04: Control the position of two lines with one variable.
- A1-05: Control the position and size of two lines with two variables.
- A1-06: Control the properties of two shapes with two variables.
- A1-07: Create a simple, regular pattern with six lines.
- A1-08: Program your pattern from Assignment 1-07 using while().
- A1-09: Draw a layered form with two new loops.
- A1-10: Redo Assignment 1-05 using mouseX and mouseY as the variables.
- A1-11: Draw two visual elements that each move in relation to the mouse in a different way.
- A1-12: Draw three visual elements that each move in relation to the mouse in a different way.
- A1-13: Move a visual element across the screen. When it disappears off the edge, move it back into the frame.
- A1-14: Draw a visual element that moves in relation to the mouse, but with a different relation when the mouse is pressed.
- A1-15: Using if and else, make the mouse perform different actions when in different parts of the window.
- A1-16: Develop a kinetic image which responds to the mouse.

Remember...

For **Thursday** this week: Theory Readings

Two presenters (you know who you are!)

Everyone else: prepare one discussion question for each reading

Man-Computer Symbiosis - J.C.R. Licklider (NMR p.73)

Personal Dynamic Media - Alan Kay & Adele Goldberg (NMR p.391)