Computation as an Expressive Medium

Lab 1: If, For, Draw, Setup and Coffee

Joshua Cuneo
What will we go over?

- Boring administrative stuff
- Variables and drawing primitives
- Instruction flow types, i.e. for, if...
- A little style
- “draw()” looping function for animation
- Simple mouse interaction
- Stop me if you haven’t covered this yet!!!
Anyone Can Program

- Just writing instructions
- I’ll make this as fun as I can
- Stop me if you get confused!
Why Do We Care?
Why Do We Care?
How This Lab Works

9:00 - 11:00

Snarky Waggle-Head Lecture

11:00 -

Workshop
TA Office Hours

- T 1-2:30, W 1-3, DM Lab.
  - + Some Fridays
  - NOT Sept. 4

- joshua.cuneo@gatech.edu
What have you gone over?

- Window size and coordinate system
- Commenting code
- Variables
- Drawing primitives
  - point
  - line
  - rect
  - ellipse
  - Triangle
- if Statements
- What else?
Variables

- Variables are placeholders for your data
- Remember middle school algebra?

```java
int i;          // declares a new variable i
i = 1;          // assigns the value of 1 to the new
                // variable
int j = i + 1;  // declares a new variable j, and
                // assigns it to i + 1, which is 2.
```
Variable Types

- **Primitives:**
  - boolean, char, byte, short, int, long, float, double

- **Objects:**
  - Built on primitives
  - Ex. String – sequence of characters
More info on drawing primitives

- RTFM!
    - Function descriptions can be really helpful once you learn how to read them...
Instruction Flow 1: Linear

- Most common flow type

  Do this
  Then do this
  Then do this
  Then do this
  etc....
Instruction Flow 2: Conditional

if something is true
  do this
otherwise, if something else is true is true
  do this
otherwise,
  do this
Instruction Flow 2: Conditional

```c
int power = 0;

if(power > 10)
{
    fire_laser();
}
else if(i > 0)       // what if this was just if?
{
    charge_laser();
}
else
{
    turn_on_laser();
}
```
Instruction Flow 3: Looping

while something is true
  do this repeatedly until that something is false
Instruction Flow 3: Looping

- Want to draw five lines, evenly-spaced apart?

```java
line(10, 10, 10, 200);
line(20, 10, 20, 200);
line(30, 10, 30, 200);
...
```

- What if you had 1000 lines to draw?
Instruction Flow 3: Looping

- Want to draw five lines, evenly-spaced apart?

```
line(10, 10, 10, 200);
line(20, 10, 20, 200);
line(30, 10, 30, 200);
...
```

What if we could run this same line of code multiple times, only changing these two numbers?
While Loops

```c
int i = 10;
while(i <= 100) {
    line(i, 10, i, 200);
    i = i + 10;
}
```

VS

```c
line(10, 10, 10, 200);
line(20, 10, 20, 200);
line(30, 10, 30, 200);
line(40, 10, 40, 200);
line(50, 10, 50, 200);
line(60, 10, 60, 200);
line(70, 10, 70, 200);
line(80, 10, 80, 200);
line(90, 10, 90, 200);
line(100, 10, 100, 200);
```

- Checks the condition before each repetition of the code.
While Loops

- Be careful of infinite loops

```java
while (true) {
    ...
}
```

```java
int i = 10;
while (i > 0) {
    line(i, 10, i, 200);
    i = i + 10;
}
```

= BAD*

*Most of the time*
These two loops are functionally the same:

```java
int i = 10;
for(int i = 10; i <= 100; i += 10)
{
    while(i <= 100)
    {
        line(i, 10, i, 200);
        i += 10;
    }
}
```

(i += 10 is equivalent to i = i + 10)
Nested Loops

```java
for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}
```

- This means that the inside loop will run all the way through, then `i` will increment, and the inside loop will run again.
Sets i to 10. That’s all for now.

for (int i = 10; i <= 100; i += 10)
{
    for (int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}
Sets j to 10. That’s all for now.

```java
for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}
```
for (int i = 10; i <= 100; i += 10)
{
    for (int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}

i's 10, j's 10. Draws a point at (10, 10). Simple enough.
for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}

Now we jump to the top of the innermost loop, and increment j by 10, and then check if it’s greater or equal to 100, which it’s not, so we continue.
Nested Loops

```java
for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}
```

i is still 10, but j is now 20. We draw the new point, then go to loop again.
for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}

Aha. So we keep looping in this j loop until it passes 100. Then what?
Nested Loops

```java
for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}
```

Well, now that the program has exited from our j loop, it sees that it’s still inside our i loop, and increments i by 10.
Nested Loops

for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}

The program reaches the j loop again, so it loops
All the way through drawing dots. Only this time, i
is 20, so the dots are further to the right.
Nested Loops

```java
for(int i = 10; i <= 100; i += 10)
{
    for(int j = 10; j <= 100; j += 10)
    {
        point(i, j);
    }
}
```

The i loop keeps incrementing in this way, drawing columns of dots, until i exceeds 100. The program then exits the code inside the i loop’s brackets.
Nested Loops

- Useful for drawing in 2D space
- What would you need for 3D space?
Function

- Just a block of code with a name
- Call the name to execute the block

```cpp
int i = 10;

while(i <= 100)
{
    line(i, 10, i, 200);
    i += 10;
}

myFunc();

void myFunc()
{
    float a = 200;
    background(255);
    a = a - 1;
    if (a < 0) {
        a = height;
    }
    line(0, a, width, a);
}
```
A Few Words About Style

- Naming conventions
  - ex. Monster AI function
    - Good: do_monster_AI()
    - Bad: snuffleupagus()

- Don't go overboard
  - this_is_my_really_cool_function_and_you_ure_an_idiot_if_you_dont_like_it()
A Few More Words About Style

- **Formatting**
  - Put curly brackets on their own line
  - Indent interior blocks of code
  - Comment your code

- **This makes your code easier to read and follow**

```c
float a = 200;

void draw()
{
    background(255);
    a = a - 1;
    if (a < 0) {
        a = height;
    }
    line(0, a, width, a);
}
```

vs

```c
float a = 200; void draw() {
    background(255);
    a = a - 1;
    if (a < 0) { a = height; }
    line(0, a, width, a); }
```
The draw() function

- Special Processing function
- Runs a number of times per second
- Used for user interaction and animation
The draw() function

Simple example:

```cpp
float a = 200;

void draw()
{
    background(255);
    a = a - 1;
    if (a < 0) {
        a = height;
    }
    line(0, a, width, a);
}
```
The draw() function

- Simple example:

```cpp
float a = 200;

void draw()
{
    background(255);
    a = a - 1;
    if (a < 0) {
        a = height;
    }
    line(0, a, width, a);
}
```

Important! The variable declaration must be outside the loop function, or else it will be reset every time draw() is called.

(Try putting it inside draw() to see what happens)
The draw() function

- Simple example:

```java
float a = 200;

void draw()
{
    background(255);
    a = a - 1;
    if (a < 0) {
        a = height;
    }
    line(0, a, width, a);
}
```

For animation, this line is necessary to clear the last frame that was drawn.

It tells Processing to paint the background of the window we see white – we could do other colors if we wanted.
The draw() function

Simple example:

```c
float a = 200;

void draw()
{
    background(255);
    a = a - 1;
    if (a < 0) {
        a = height;
    }
    line(0, a, width, a);
}
```

This is the meat of what makes the line move. Each loop, we subtract 1 from the variable `a`, which we later use when drawing the line, so it will appear to move.

When `a` is 0, we’ve hit the top of the window, and reset `a` to the highest value.
The setup() function

- Used to define window properties at startup
  - size(width, height): size of drawing window (in pixels)
  - framerate(fps): number of times per second that draw() is called

```java
void setup()
{
  size(640, 480);
  framerate(30);
}
```
Tracking Mouse Position

- Special Processing variables
  - `mouseX`, `mouseY`: current X and Y of cursor, relative to the top-left of drawing window.

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

A simple program that moves a line left or right to follow your cursor. Note that it stops tracking your mouse when it’s not in the drawing window.
Reacting to Mouse Clicks

- **Use the variable `mousePressed`.**

```cpp
void draw() {
  if (mousePressed) {
    point(mouseX, mouseY);
  }
}
```

This simple function will draw a point in the window whenever the mouse is in the clicked position. Try clicking and dragging around the window and see what happens.

Can you think of a way to make the drawing smoother?

- **Also see `mouseReleased`, `mouseMoved`, and `mouseDragged`.**
width and height

- width = width of the screen
- height = height of the screen
Assignment 1

- Questions?