Design Programming

DECO1012 & DECO2011
Rob Saunders
Rob Saunders

web: http://www.arch.usyd.edu.au/~rob

e-mail: rob@arch.usyd.edu.au

office: Room 274, Wilkinson Building
Animation & Functions
Continuous Drawing
The **draw()** Function

- Programs that run continuously must include a **draw()** function.
- The code inside a **draw()** block runs until the stop button is pressed or the window is closed.

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>void draw() {</td>
<td>1</td>
</tr>
<tr>
<td>println(frameCount);</td>
<td>2</td>
</tr>
<tr>
<td>}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>
Frames

- The `frameRate()` function changes the maximum number of frames per second.
- The function controls only the maximum frame rate—it can not speed up a program that runs slowly because of equipment limitations.
- The `frameCount` variable contains the total number of frames displayed.
Continuous Drawing

```c
float y = 0.0;

void draw() {
    line(0, y, 100, y);
    y = y + 0.5;
}
```

```
y = 10
```

```
y = 50
```

```
y = 80
```
Unrolling the **draw()** Loop

```cpp
float y = 0.0
line(0, 0.0, 100, 0.0)
y = 0.0 + 0.5
line(0, 0.5, 100, 0.5)
y = 0.5 + 0.5
line(0, 1.0, 100, 1.0)
y = 1.0 + 0.5
```

Etc...
Clearing the Background

float y = 0.0;

void draw() {
    background(204);
    line(0, y, 100, y);
    y = y + 0.5;
}

y = 10
y = 50
y = 80
Animating the Background

float y = 0.0;

void draw() {
    background(y * 2.5);
    line(0, y, 100, y);
    y = y + 0.5;
}

y = 10
y = 50
y = 80
Using Conditionals to Loop Animations

float y = 0.0;

void draw() {
    background(204);
    line(0, y, 100, y);
    y = y + 0.5;
    if (y > height) {
        y = 0.0;
    }
}

y = 10
y = 50
y = 80
Using `setup()` and `draw()`

- Each program can have only one `setup()` and one `draw()`.
  - Code outside of `setup()` and `draw()` is run.
  - Code inside the `setup()` block is run once.
  - Code inside the `draw()` block is run continuously until the program is stopped.
int y = 0;

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

void draw() {
    line(0, y, 300, y);
    y = y + 4;
}

The area between { and } is a block

Return value

Function

Data type

Assignment operator

Statement terminator

Parameters

Expression
Separating Setting Up from Animation

```c
float y = 0.0;

void setup() {
    size(100, 100);
    smooth();
    fill(0);
}

void draw() {
    background(204);
    ellipse(50, y, 70, 70);
    y += 0.5;
    if (y > height + 50.0) {
        y = -50.0;
    }
}
```
Unrolling \texttt{setup()} and \texttt{draw()}

float \texttt{y} = 0.0 
\texttt{size(100, 100)}

\texttt{smooth()} 
\texttt{fill(0)} 
\texttt{background(204)}

\texttt{ellipse(50, 0.0, 70, 70)}
\texttt{y = 0.5}
\texttt{background(204)}

\texttt{ellipse(50, 0.5, 70, 70)}
\texttt{y = 1.0}
\texttt{background(204)}

\texttt{ellipse(50, 1.0, 70, 70)}
\texttt{y = 1.5}
\texttt{background(204)}

\texttt{ellipse(50, 1.5, 70, 70)}
\texttt{y = 2.0}
\texttt{background(204)}

\texttt{ellipse(50, 2.0, 70, 70)}
\texttt{y = 2.5}
\texttt{background(204)}

\texttt{ellipse(50, 2.5, 70, 70)}
\texttt{y = 3.0}
\texttt{background(204)}

\texttt{ellipse(50, 3.0, 70, 70)}
\texttt{y = 3.5}
\texttt{background(204)}

\texttt{ellipse(50, 3.5, 70, 70)}
\texttt{y = 4.0}
\texttt{background(204)}

\texttt{ellipse(50, 4.0, 70, 70)}
\texttt{y = 4.5}
\texttt{background(204)}

\texttt{ellipse(50, 4.5, 70, 70)}
\texttt{y = 5.0}
\texttt{background(204)}

\texttt{ellipse(50, 5.0, 70, 70)}
\texttt{y = 5.5}
\texttt{background(204)}

\texttt{ellipse(50, 5.5, 70, 70)}
\texttt{y = 6.0}
\texttt{background(204)}

\texttt{ellipse(50, 6.0, 70, 70)}
\texttt{y = 6.5}
\texttt{background(204)}

\texttt{ellipse(50, 6.5, 70, 70)}
\texttt{y = 7.0}
\texttt{background(204)}

\texttt{ellipse(50, 7.0, 70, 70)}
\texttt{y = 7.5}
\texttt{background(204)}

\texttt{ellipse(50, 7.5, 70, 70)}
\texttt{y = 8.0}
\texttt{background(204)}

\texttt{ellipse(50, 8.0, 70, 70)}
\texttt{y = 8.5}
\texttt{background(204)}

\texttt{ellipse(50, 8.5, 70, 70)}
\texttt{y = 9.0}
\texttt{background(204)}

\texttt{ellipse(50, 9.0, 70, 70)}
\texttt{y = 9.5}
\texttt{background(204)}

\texttt{ellipse(50, 9.5, 70, 70)}
\texttt{y = 10.0}
\texttt{background(204)}

Etc.
noLoop() 

- The noLoop() function stops your sketch from calling draw().
  - Using noLoop() you can inspect your animation at specific points, e.g., the first frame.

- It is possible to start a frozen animation using the loop() function.
  - NOTE: The loop() function needs to be called from a function that responds to something outside of the animation. We’ll get to that later...
Using **noLoop()** to Inspect an Animation

float y = 0.0;

void setup() {
  smooth();
  noStroke();
  colorMode(HSB);
  noLoop();
}

void draw() {
  fill(frameCount % 255, 100, 100);
  ellipse(50, y, 70, 70);
  y += 0.5;
}
Using `noLoop()` to Inspect an Animation

```java
float y = 0.0;

void setup() {
    smooth();
    noStroke();
    colorMode(HSB);
}

void draw() {
    fill(frameCount % 255, 100, 100);
    ellipse(50, y, 70, 70);
    y += 0.5;
    if (y == 40) { noLoop(); }
}
```
Variable Scope

int d = 51; // Global variable d can be used everywhere

void setup() {
  size(100, 100);
  int val = d * 2; // Local variable val can only be used in setup()
  fill(val);
}

void draw() {
  int y = 60; // Local variable y can only be used in draw()
  line(0, y, d, y);
  y -= 25;
  line(0, y, d, y);
}
Example Scope Errors

```java
void draw() {
    int d = 80;    // This variable can be used everywhere in draw()
    if (d > 50) {
        int x = 10;  // This variable can be used only in this if block
        line(x, 40, x+d, 40);
    }
    line(0, 50, d, 50);
    line(x, 60, x+d, 60); // ERROR! x can't be read outside block
}
```

```java
void draw() {
    for (int y = 20; y < 80; y += 6) { // The variable y can be used
        line(20, y, 50, y);          // only within the for block
    }
    line(y, 0, y, 100); // ERROR! y can't be accessed outside for
}
```
Functions
Functions

- A function is a self-contained module of code that can be re-used.
- You’ve been using the functions included with Processing such as `size()`, `line()`, `stroke()`, and `translate()` to write your programs, but it’s also possible to write your own functions that make a program modular.
Functions

- Functions make code more concise by extracting the common elements and making them into code blocks that can be run many times within the program.
- This makes the code easier to read and update and reduces the chance of errors.
Functions and Parameters

- Functions often have parameters that modify their actions.
  - The `line()` function has four parameters that define the end points.

- Functions can be defined with different numbers of parameters.
  - A single parameter to the `fill()` function defines a grey value and three parameters define an RGB colour.
Creating a Function

```cpp
void setup() {
  size(100, 100);
  noStroke();
  smooth();
  noLoop();
}

void draw() {
  fill(255);
  ellipse(50, 50, 60, 60);
  fill(0);
  ellipse(50+10, 50, 30, 30);
  fill(255);
  ellipse(50+16, 45, 6, 6);
}
```
Creating a Function

void setup() {
  size(100, 100);
  noStroke();
  smooth();
  noLoop();
}

void draw() {
  eye();
}

void eye() {
  fill(255);
  ellipse(50, 50, 60, 60);
  fill(0);
  ellipse(50+10, 50, 30, 30);
  fill(255);
  ellipse(50+16, 45, 6, 6);
}
Creating a Function

```cpp
void setup() {
    size(100, 100);
    noStroke();
    smooth();
    noLoop();
}

void draw() {
    eye(65, 44);
    eye(20, 50);
}

void eye(int x, int y) {
    fill(255);
    ellipse(x, y, 60, 60);
    fill(0);
    ellipse(x+10, y, 30, 30);
    fill(255);
    ellipse(x+16, y-5, 6, 6);
}
```
void setup() {
    size(100, 100);
    smooth();
    noLoop();
}

void draw() {
    drawX();
}

void drawX() {
    stroke(160);
    strokeWeight(20);
    line(0, 5, 60, 65);
    line(60, 5, 0, 65);
}
Creating a Function

```cpp
void setup() {
    size(100, 100);
    smooth();
    noLoop();
}

void draw() {
    drawX(0);
}

void drawX(int gray) {
    stroke(gray);
    strokeWeight(20);
    line(0, 5, 60, 65);
    line(60, 5, 0, 65);
}
```
Creating a Function

```cpp
void setup() {
    size(100, 100);
    smooth();
    noLoop();
}

void draw() {
    drawX(0, 30, 40, 30, 36);
}

void drawX(int gray, int weight, int x, int y, int size) {
    stroke(gray);
    strokeWeight(weight);
    line(x, y, x+size, y+size);
    line(x+size, y, x, y+size);
}
```
Creating a Function

```java
void draw() {
    for (int i = 0; i < 70; i++) {
        drawX(
            int(random(255)), // gray
            int(random(30)),  // width
            int(random(width)), // x
            int(random(height)), // y
            100                  // size
        );
    }
}
```
Abstraction

void draw() {
    leaf(26, 83, 60, 1);
}

void leaf(float x, float y, float sizeX, float sizeY) {
    pushMatrix();
    translate(x, y);     // Move to position
    scale(sizeX, sizeY); // Scale to size
    beginShape();        // Draw the shape
    vertex(1.0, -0.7);
    bezierVertex(1.0, -0.7, 0.4, -1.0, 0.0, 0.0);
    bezierVertex(0.0, 0.0, 1.0, 0.4, 1.0, -0.7);
    endShape();
    popMatrix();
}
void draw() {
    vine(33, 9, 16);
}

void vine(int x, int numLeaves, int leafSize) {
    stroke(255);
    line(x, 0, x, height);
    noStroke();
    int gap = height / numLeaves; // Calculate average leaf gap
    int leafSizeX = leafSize;
    int leafSizeY = leafSize;
    for (int i = 0; i < numLeaves; i++) {
        leaf(x, i * gap + random(gap), leafSizeX, leafSizeY);
        leafSizeX = -leafSizeX; // Flip leaf direction
    }
}
// Draws an X with the given colour, thickness, position and size
void drawX(color col, float weight, float x, float y, float s) {
    stroke(col);
    strokeWeight(weight);
    line(x-s/2, y-s/2, x+s/2, y+s/2);
    line(x+s/2, y-s/2, x-s/2, y+s/2);
}
Function Overloading

```java
void draw() {
    translate(30, 30);
    // Draw thick white X at origin with size 60
    drawX(color(255), 20, 0, 0, 60);
    // Draw thinner purple X at origin with size 60
    drawX(color(191, 63, 191), 10, 0, 0, 60);
    // Draw thin black X at origin with size 60
    drawX(color(0), 1, 0, 0, 60);
    // Draw 2-pixel blue X at (44, 48) with size 36
    drawX(color(32, 32, 191), 2, 44, 48, 36);
}
```
Function Overloading

// Draw an X with default values
void drawX() {
    drawX(0, 1, 0, 0, 60);
}

// Draw an X with the gray value and thickness set by parameter
void drawX(int gray, float weight) {
    drawX(color(gray), weight, 0, 0, 60);
}

// Draw an X with the RGB colour and thickness set by parameter
void drawX(int r, int g, int b, float weight) {
    drawX(color(r, g, b), weight, 0, 0, 60);
}
void draw() {
    translate(30, 30);
    // Draw thick white X
    drawX(255, 20);
    // Draw thinner purple
    drawX(191, 63, 191, 10);
    // Draw an X
    drawX();
    // Draw thin blue X at (44, 48) with size 36
    drawX(color(32, 32, 191), 2, 44, 48, 36);
}
Drawing Static Sketches Using Functions

smooth();
noStroke();
translate(30, 30);
drawX(color(255), 20, 0, 0, 60);

void drawX(color col, float weight, float x, float y, float s) {
    stroke(col);
    strokeWeight(weight);
    line(x-s/2, y-s/2, x+s/2, y+s/2);
    line(x+s/2, y-s/2, x-s/2, y+s/2);
}
void setup() {
    smooth();
    noStroke();
    translate(30, 30);
    drawX(color(255), 20, 0, 0, 60);
}

void drawX(color col, float weight, float x, float y, float s) {
    stroke(col);
    strokeWeight(weight);
    line(x-s/2, y-s/2, x+s/2, y+s/2);
    line(x+s/2, y-s/2, x-s/2, y+s/2);
}
void setup() {
  smooth();
  noStroke();
  noLoop();
}

void draw() {
  translate(30, 30);
  drawX(color(255), 20, 0, 0, 60);
}

void drawX(color col, float weight, float x, float y, float s) {
  stroke(col);
  strokeWeight(weight);
  line(x-s/2, y-s/2, x+s/2, y+s/2);
  line(x+s/2, y-s/2, x-s/2, y+s/2);
}
Returning Values

- Data output from a function is called the return value.
  - All functions are expected to return a value, such as an `int` or a `float`.
- If the function does not return a value, the special type `void` is used.
Returning Values

- The keyword `return` is used to exit a function and return to the point in the program from which it was called.
- When a function outputs a value, `return` is used to specify what value should be returned.
- The `return` statement is typically the last line of a function because functions exit immediately after a return.
Returning Values

- We’ve already been using functions that return values.
  - For example, `random()` returns a `float`, `int()` returns an `int`, and the `color()` function returns a value of the `color` data type.
Using Functions that Return Values

- If a function returns a value, the function almost always appears to the right of an assignment operator or as a part of a larger expression.
- A function that does not return a value is often used as a complete statement.
- If a function’s return value is not used immediately, or assigned to a variable, the value will be lost.
Using Functions that Return Values

Return values can be stored in variables for later use

```cpp
float d = random(0, 100);
ellipse(50, 50, d, d);
```

Variables used to store return values need to be of the correct type...

```cpp
int d = random(0, 100);    // ERROR! random() returns floats
ellipse(50, 50, d, d);
```

...or we need to use other functions to convert return values to the type of the variable

```cpp
int d = int(random(0, 100));    // int() converts the float value
ellipse(50, 50, d, d);
```
Writing Your Own Functions that Return Values

Source Code

```java
void setup() {
    float f = average(12.0, 6.0);
    println(f);
}

float average(float num1, float num2) {
    float ave = (num1 + num2) / 2.0;
    return ave;
}
```

Output

9.0
void setup() {
    float c = fahrenheitToCelsius(451.0);
    println(c);
}

float fahrenheitToCelsius(float t) {
    float f = (t-32.0) * (5.0/9.0);
    return f;
}
Lab Exercises
Lab Exercises

- Create an animation that moves a shape from left to right across the screen.
  - When the shape moves off the right edge, return its position to the left by resetting the value that controls its x-position.

- Create a 2nd animation, based on the first, that moves the shape from right to left.
  - Try changing your animation so that the shape bounces back-and-forth across the screen by controlling the direction of motion.
Lab Exercises

- Write a function to draw a shape.
  - Use the function to draw the shape to the screen multiple times at different positions.
  - Extend the function by adding parameters to control additional aspects of its drawing.

- Write a function to use with a `for` structure to create a pattern evoking a liquid substance.
  - Consider using a function to draw sine waves.