

Variables and Control Structures

CS 110

Eric Eaton

Review

- Random numbers
- mouseX, mouseY
- setup() & draw()
- frameRate(), loop(), noLoop()
- Mouse and Keyboard interaction
- Arcs, curves, bézier curves, custom shapes
- Hue-Saturation-Brightness vs. Red-Green-Blue color
- Decimal, Hex, Binary numbers and colors
- Example Sketches
- OpenProcessing website

Variables

- A name to which data can be assigned
- A variable name is declared as a specific data type
- Names must begin with a letter, “_” or “\$” and can contain letters, digits, “_” and “\$”

```
boolean bReady = true;
int i;
int j = 12;
float fSize = 10.0;
color _red = color(255,0,0);
String name123 = "Fred";
PImage img;
```

Variable Uses

- Use a value throughout your program,
 - but allow it to be changed
- As temporary storage for a intermediate computed result
- ... etc

Primitive Data Types

Type	Range	Default	Bytes
boolean	{ true, false }	false	?
byte	{ 0..255 }	0	1
int	{ -2,147,483,648 .. 2,147,483,647 }	0	4
long	{ -9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807 }	0	8
float	{ -3.40282347E+38 .. 3.40282347E+38 }	0.0	4
double	<i>much larger/smaller</i>	0.0	8
color	{ #00000000 .. #FFFFFFFF }	<i>black</i>	4
char	<i>a single character 'a', 'b', ...</i>	'\u0000'	2

Data Type Conversion

- Some variable types can be converted to other types
- Via **casting** (from Java, the foundation for Processing)

```
float f = 10.0;  
int i = (int) f;
```

- Processing builds in additional type conversion functions:

```
// binary(...), boolean(...), byte(...),  
// char(...), float(...), str(...)
```

```
float f = 10.0;  
int i;
```

```
//i = f; // Throws a runtime error  
i = int(f);
```

```
println( char(65) ); // Prints the character 'A'
```

Other "things" ...

Type	Range	Default	Bytes
String	a series of chars in quotes "abc"	null	?
PImage	an image	null	?
PFont	a font for rendering text	null	?

...

```
String message = "Hello World!";
```

Conditionals: if-statement

Programmatic branching ...

```
if ( boolean_expression ) {  
    statements;  
}
```

```
// What does this do?
```

```
void draw() {  
    if ( mouseX > 50 && mouseY > 50 ) {  
        ellipse( mouseX, mouseY, 10, 10 );  
    }  
}
```


Relational Expressions

- < less than
- > is greater than
- <= is less than or equal to
- >= is greater than or equal to
- == is equivalent
- != is not equivalent

Relational Expressions: Examples

1. `if (true) { ... }`
2. `if (10 > 10) { ... }`
3. `if (10 >= 10) { ... }`
4. `if ('a' == 'a') { ... }`
5. `if ('a' != 'a') { ... }`
6. `if ("Bryn Mawr" != "bryn mawr") { ... }`

Logical Expressions

&& logical conjunction (and)

- both expressions must be true for conjunction to be true

|| logical disjunction (or)

- either expression must be true for disjunction to be true

! logical negation (not)

- true \rightarrow false, false \rightarrow true

Logical Expression Examples

1. `if ((2 > 1) && (3 > 4)) { ... }`
2. `if (("blah" == "blah") && (1 + 2 == 3)) { ... }`
3. `if (!false) { ... }`
4. `if (!(1 < -1)) { ... }`
5. `if (!(10 < 20) || false) { ... }`
6. `if (!(10 > 20) && (10 < 20)) { ... }`
7. `if ((true || false) && true) { ... }`
8. `if ((true && false) || true) { ... }`
9. ...

Conditionals: if-else-statement

```
if ( boolean_expression ) {  
    statements executed when boolean_expression is true;  
} else {  
    statements executed when boolean_expression is false;  
}
```

```
// What does this do?  
void draw() {  
    if ( mouseY < 50 ) {  
        println("the sky");  
    } else {  
        println("the ground");  
    }  
}
```

Conditionals: if-else-if-statement

```
if ( boolean_expression_1 ) {  
    statements;  
} else if ( boolean_expression_2 ) {  
    statements;  
} else if ( boolean_expression_3 ) {  
    statements;  
} else {  
    statements;  
}
```

```
void setup() {
  size(500,500);
  smooth();
  ellipseMode(CENTER);
}

void draw() {
  if (mouseX < 250) {
    stroke(255, 0, 0);

    if (mouseY < 250) {
      fill(0, 255, 0);
    } else {
      fill(0, 0, 255);
    }

  } else {
    stroke(0, 0, 255);

    if (mouseY < 250) {
      fill(255, 0, 0);
    } else {
      fill(255);
    }
  }
  ellipse(mouseX, mouseY, 50, 30);
}
```

What will this do?

```
void setup() {  
  size( 500, 500 );  
  smooth();  
}
```

```
void draw() {  
  
  if ( mouseX > 100 )  
  {  
    background( 255, 0, 0 );  
  } else if ( mouseX > 200 )  
  {  
    background( 0, 0, 255 );  
  }  
  
}
```

What does this do?

What does this do?

```
void setup() {  
  size( 500, 500 );  
  smooth();  
}  
  
void draw() {  
  
  if ( mouseX > 200 )  
  {  
    background( 255, 0, 0 );  
  }  
  
  if ( mouseX > 100 )  
  {  
    background( 0, 0, 255 );  
  }  
  
}
```

Conditionals: switch-statement

- Works like a if-else statement.
- Convenient for large numbers of value tests.

```
switch( expression ) {  
    case label1:           // label1 equals expression  
        statements;  
        break;  
    case label2:           // label2 equals expression  
        statements;  
        break;  
    default:               // Nothing matches  
        statements;  
}
```

```
void setup() {  
  size(500, 500);  
  smooth();  
}  
  
void draw() {}  
  
void keyPressed() {  
  switch(key)  
  {  
    case 'l':  
    case 'L':  
      println("Turning left");  
      break;  
    case 'r':  
    case 'R':  
      println("Turning right");  
      break;  
  }  
}
```

What does this do?

```
int positionX = 250;
int positionY = 250;
int deltaX = 0;
int deltaY = 0;

void setup() {
  size(500, 500);
  smooth();
}

void draw() {
  background(255);

  positionX = positionX + deltaX;
  positionY = positionY + deltaY;

  if (positionX < 0)
    positionX = 0;
  if (positionX > width)
    positionX = width;
  if (positionY < 0)
    positionY = 0;
  if (positionY > height)
    positionY = height;

  ellipse(positionX, positionY, 50, 50);
}
```

```
void keyPressed() {
  switch (keyCode) {
  case 37:
    deltaX = -2;
    deltaY = 0;
    break;
  case 39:
    deltaX = 2;
    deltaY = 0;
    break;
  case 38:
    deltaY = -2;
    deltaX = 0;
    break;
  case 40:
    deltaY = 2;
    deltaX = 0;
    break;
  case 32:
    deltaX = 0;
    deltaY = 0;
    break;
  }
}
```

Equations of Motion (Simplified)

s = displacement

t = time

v = velocity

a = acceleration

- Constant acceleration (a)

$$s_{i+1} = s_i + v_i \Delta t$$

$$v_{i+1} = v_i + a \Delta t$$

```
float sx = 0.0;    // x position
float sy = 0.0;    // y position
float vx = 1.0;    // x velocity
float vy = 1.0;    // y velocity
float ay = 0.2;    // y acceleration (gravity)
```

```
void setup() {
  size(500, 500);
  smooth();
  ellipseMode(CENTER);
}
```

```
void draw() {
  sx = sx + vx;
  sy = sy + vy;
  vy = vy + ay;

  if (sx <= 0.0 || sx >= width) vx = -vx;
  if (sy >= (height-10.0)) vy = -0.9*vy;

  background(255);
  ellipse(sx, sy, 20, 20);
}
```

What does this do?

Iteration

Repetition of a program block

- Iterate when a block of code is to repeated multiple times.

Options

- The while-loop
- The for-loop

Iteration: while-loop

```
while ( boolean_expression ) {  
    statements;  
    // continue;  
    // break;  
}
```

- Statements are repeatedly executed while the boolean expression remains true;
- To break out of a while loop, call **break**;
- To stop execution of statements and start again, call **continue**;
- All iterations can be written as while-loops.


```
void setup() {
  size(500, 500);
  smooth();

  float diameter = 500.0;
  while ( diameter > 1.0 ) {
    ellipse( 250, 250, diameter, diameter);
    diameter = diameter * 0.9;
  }
}

void draw() { }
```

What does this do?

```
void setup() {
  size(500, 500);
  smooth();

  float diameter = 500.0;
  while ( true ) {
    ellipse( 250, 250, diameter, diameter);
    diameter = diameter * 0.9;
    if (diameter <= 1.0 ) break;
  }
}

void draw() { }
```

An aside ... Operators

`+`, `-`, `*`, `/` and ...

`i++;` *equivalent to*

`i = i + 1;`

`i += 2;` *equivalent to*

`i = i + 2;`

`i--;` *equivalent to*

`i = i - 1;`

`i -= 3;` *equivalent to*

`i = i - 3;`

`i *= 2;` *equivalent to*

`i = i * 2;`

`i /= 4;` *equivalent to*

`i = i / 4;`

`i % 3;` the remainder after `i` is divided by 3 (modulo)

Example

...

```
factorial = 1;
```

```
while ( myNumber > 0 ) {
```

```
    factorial *= myNumber;
```

```
    --myNumber;
```

```
}
```

```
return factorial;
```

The 3 Parts of a Loop

```
...
int i = 1 ;           ← initialization of loop control variable

// count from 1 to 100
while ( i < 101 ) {  ← test of loop termination condition
    System.out.println( i );
    i = i + 1 ;      ← modification of loop control variable
}
return 0 ;
}
```

The for Loop Repetition Structure

- The **for** loop handles details of the counter-controlled loop “automatically”.
- The initialization of the the loop control variable, the termination condition test, and control variable modification are handled in the **for** loop structure.

```
for (int i = 1; i < 101; i = i + 1) {  
    ↑  
    initialization  
    ↑  
    test  
    ↑  
    modification  
}
```

for Loop Examples

- A *for* loop that counts from 0 to 9:

```
// modify part can be simply "i++"  
for ( i = 0; i < 10; i = i + 1 ) {  
    System.out.println( i ) ;  
}
```

- ...or we can count backwards by 2' s :

```
// modify part can be "i -= 2"  
for ( i = 10; i > 0; i = i - 2 ) {  
    System.out.println( i ) ;  
}
```

When Does a *for* Loop Initialize, Test and Modify?

- Just as with a while loop, a for loop
 - initializes the loop control variable before beginning the first loop iteration
 - performs the loop termination test before each iteration of the loop
 - modifies the loop control variable at the very end of each iteration of the loop
- The for loop is easier to write and read for counter-controlled loops.

```
void setup() {  
  size(500, 500);  
  smooth();  
  
  float diameter = 500.0;  
  while ( diameter > 1.0 ) {  
    ellipse( 250, 250, diameter, diameter);  
    diameter = diameter - 10.0;  
  }  
}
```

```
void draw() { }
```

```
void setup() {  
  size(500, 500);  
  smooth();  
  
  for (float diameter = 500.0; diameter > 1.0; diameter -= 10.0 )  
  {  
    ellipse( 250, 250, diameter, diameter);  
  }  
}
```

```
void draw() { }
```



```
void mousePressed() {  
  
    for (int i = 0; i < 10; i++ )  
    {  
        print( i );  
    }  
    println();  
  
}  
  
void draw() { }
```

```
void mousePressed() {  
    for (int i = 0; i < 10; i++ )  
    {  
        if ( i % 2 == 1 ) continue;  
        print( i );  
    }  
    println();  
}  
  
void draw() { }
```

```
// for2
float delta = 5.0;
float factor = 0.0;

void setup() {
  size(500, 500);
}

void draw() {

  factor+=0.2;
  noStroke();

  for (float r=0.0; r<height; r+=delta) {
    for (float c=0.0; c<width; c+=delta) {

      // Use factor to scale shape
      float x = map(c, 0.0, 500.0, 0.0, 3.0*TWO_PI);
      float y = map(r, 0.0, 500.0, 0.0, 3.0*TWO_PI);
      float shade = map(sin(factor)*sin(x)*sin(y), -1.0, 1.0, 0, 255);

      // Use factor to shift shade
      // float x = map(c, 0.0, 500.0, factor, factor+3.0*TWO_PI);
      // float y = map(r, 0.0, 500.0, factor, factor+3.0*TWO_PI);
      // float shade = map(sin(x)*sin(y), -1.0, 1.0, 0, 255);

      fill( shade );
      rect(r, c, delta, delta);
    }
  }
}
```

The *break* & *continue* Statements

- The `break` & `continue` statements can be used in **while** and **for** loops to cause the remaining statements in the body of the loop to be skipped; then:
 - `break` causes the looping itself to abort, while...
 - `continue` causes the next turn of the loop to start. In a **for** loop, the modification step will still be executed.

Example break in a for Loop

```
...
int i;
for (i = 1; i < 10; i = i + 1) {
    if (i == 5) {
        break;
    }
    System.out.println(i);
}
System.out.println("\nBroke out of loop at i = " + i);
```

•OUTPUT:

• 1 2 3 4

• **Broke out of loop at i = 5.**

Example continue in a for Loop

```
...
int i;
for (i = 1; i < 10; i = i + 1) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
}
System.out.println("Done");
```

OUTPUT:

1 2 3 4 6 7 8 9

Done.

Problem: continue in while Loop

```
// This seems equivalent to for loop
// in previous slide—but is it??
...
int i = 1;
while (i < 10) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
    i = i + 1;
}
System.out.println("Done");
```

OUTPUT:

???

Variable Scope

Variable scope:

- That set of code statements in which the variable is known to the compiler
- Where it can be referenced in your program.
- Limited to the ***code block*** in which it is defined.
 - A ***code block*** is a set of code enclosed in braces (`{ }`).

One interesting application of this principle allowed in Java involves the **for loop** construct.

for-loop index

- Can declare and initialize variables in the heading of a **for loop**.
- These variables are local to the for-loop.
- They may be reused in other loops.

```
String s = "hello world";  
int count = 1;  
for (int i = 0; i < s.length(); i++)  
{  
    count *= 2;  
}  
//using 'i' here generates a compiler error
```


Comments

- **Line comment**

- Begins with the symbols `//`
- Compiler ignores remainder of the line
- Used for the coder or for a programmer who modifies the code

```
    if (birthYear > currentYear) // birth year is invalid
        then . . .
```

- **Block comment**

- Begins with `/*` and ends with `*/`
- Compiler ignores anything in between
- Can span several lines
- Provides documentation for the users of the program

```
/* File: Date
   Author: Joe Smith
   Date: 9/1/09
*/
```

text()

- Strings can be drawn on a sketch using the text() function.
- Can set text position, font, size, alignment, ...
- Font files are loaded from the data folder.

```
// Set attributes
textSize( sizeInPixels );
textAlign( {LEFT | CENTER | RIGHT}
           [, {TOP, BOTTOM, CENTER, BASELINE}] );
fill( color );

// Render text
text( string, X, Y );
text( string, X, Y, width, height );
```

```
// text
void setup() {
  size(500, 500);
  noLoop();
}

void draw() {
  // bounding box
  stroke(0);
  fill(255);
  rect(50, 50, 400, 400);

  // text options
  fill(0); // black text
  text("Default", 50, 50, 400, 400);
  textAlign(CENTER);
  text("CENTER", 50, 50, 400, 400);
  textAlign(RIGHT);
  text("RIGHT", 50, 50, 400, 400);
  textAlign(CENTER, CENTER);
  text("CENTER-CENTER", 50, 50, 400, 400);
  textAlign(RIGHT, BOTTOM);
  text("RIGHT-BOTTOM", 50, 50, 400, 400);
  textAlign(LEFT, BOTTOM);
  text("LEFT-BOTTOM", 50, 50, 400, 400);
}
```