CS 106A, Lecture 7
Booleans, Control Flow and Scope

suggested reading:

Java Ch. 4

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

Announcements

- 2 Handouts (“Methods” online , Section handout in hardcopy)
- *Reminder:* Assignment 2 YEAH (Your Early Assignment Help) Hours **tonight 7-8PM in 320-105**.
- Permanent section change deadline **tomorrow 10/10 at 5PM.**

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

Expressions

- You can combine literals or variables together into **expressions** using binary operators:

+	Addition	*	Multiplication
–	Subtraction	/	Division
		%	Remainder

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `*  /  %` have a higher level of precedence than `+  -`

```
1 + 3 * 4        is 13

6 + 8 / 2 * 3
6 +   4   * 3
6 +     12       is 18
```

  - Parentheses can alter order of evaluation, but spacing does not:

```
(1 + 3) * 4      is 16
1+3 * 4-2        is 11
```

Integer division

- When we divide integers, the quotient is also an integer.

  `14 / 4` is `3`, not `3.5`. *(Java ALWAYS rounds down.)*

```
      3              4              52
4 ) 14        10 ) 45       27 ) 1425
    12              40            135
     2               5             75
                                   54
                                   21
```

- More examples:
  - `32 / 5`   is `6`
  - `84 / 10`   is `8`
  - `156 / 100` is `1`

  - Dividing by `0` causes an error when your program runs.

Type Casting

- Type casting makes the computer treat one type as another for one operation.

```
int x = 1;
int y = x / 2;  // 0!
----
int x = 1;
double y = (double)x / 2;   // 0.5
// or
double y = x / 2.0;         // 0.5
----
double y = 3.5;
int x = (int)y;             // 3 -> truncation!
```

Shorthand Operators

Shorthand	Equivalent longer version
`variable += value;`	*`variable = variable + value;`*
`variable -= value;`	*`variable = variable - value;`*
`variable *= value;`	*`variable = variable * value;`*
`variable /= value;`	*`variable = variable / value;`*
`variable %= value;`	*`variable = variable % value;`*
`variable++;`	***`variable = variable + 1;`***
`variable--;`	***`variable = variable – 1;`***

```
x += 3;             // x = x + 3;
number *= 2;        // number = number * 2;
x++;                // x = x + 1;
```

Practice

- `1 / 2`
- `1.0 / 2`
- `2 + 2 / 3`
- `2 + (double)1 / 2`
- `(2 + 2) / 3`

Constants

- **constant**: A variable that cannot be changed after it is initialized. Declared at the top of your class, *outside of the run() method*. Can be used anywhere in that class.
- Better style – can easily change their values in your code, and they are easier to read in your code.
- Syntax:

```
private static final type name = value;
```

  – name is usually in ALL_UPPER_CASE

  – Examples:

```
private static final int DAYS_IN_WEEK = 7;
private static final double INTEREST_RATE = 3.5;
private static final int SSN = 658234569;
```

Booleans

1 < 2

Booleans

$1 < 2$

```
true
```

Relational Operators

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`true`
`!=`	does not equal	`3.2 != 2.5`	`true`
`<`	less than	`10 < 5`	`false`
`>`	greater than	`10 > 5`	`true`
`<=`	less than or equal to	`126 <= 100`	`false`
`>=`	greater than or equal to	`5.0 >= 5.0`	`true`

* All have equal precedence

Relational Operators

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`true`
`!=`	does not equal	`3.2 != 2.5`	`true`
`<`	less than	`10 < 5`	`false`
`>`	greater than	`10 > 5`	`true`
`<=`	less than or equal to	`126 <= 100`	`false`
`>=`	greater than or equal to	`5.0 >= 5.0`	`true`

* All have equal precedence

Compound Expressions

In order of precedence:

Operator	Description	Example	Result
`!`	not	`!(2 == 3)`	`true`
`&&`	and	`(2 == 3) && (-1 < 5)`	`false`
`||`	or	`(2 == 3) || (-1 < 5)`	`true`

Cannot "chain" tests as in algebra; use && or || instead

```
// assume x is 15
2 <= x <= 10
true   <= 10
Error!
```

```
// correct version
2 <= x && x <= 10
true   && false
false
```

Precedence Madness

Precedence: arithmetic > relational > logical

```
5 * 7 >= 3 + 5 * (7 - 1) && 7 <= 11
5 * 7 >= 3 + 5 * 6 && 7 <= 11
35    >= 3 + 30 && 7 <= 11
35    >= 33 && 7 <= 11
true && true
true
```

Boolean Variables

```
// Store expressions that evaluate to true/false
boolean x = 1 < 2;             // true
boolean y = 5.0 == 4.0;        // false
```

Boolean Variables

```java
// Store expressions that evaluate to true/false
boolean x = 1 < 2;            // true
boolean y = 5.0 == 4.0;       // false

// Directly set to true/false
boolean isMonday = true;
boolean isRaining = false;
```

Short-Circuit Evaluation

- Stop evaluating a boolean expression as soon as we know the answer.

```
// ??? doesn't matter
boolean p = TRUE || ???;


boolean p = FALSE && ???;
```

Short-Circuit Evaluation

- Stop evaluating a boolean expression as soon as we know the answer.

```
// regardless of (4 <= 2), p is always true!
boolean p = (5 > 3) || (4 <= 2);

// avoid division by 0 if x is zero
boolean p = (x != 0) && (y / x == 0);
```

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

Aside: Strings

- **String** is another type of variable that stores text.

```
String str = “hello there”;
```

- You put **String**s inside the parentheses of println to print that text.

```
println(str);          // hello there
```

- Strings can be *concatenated* using +.

```
String str = “hello”;
println(str + ” CS106A!”);    // hello CS106A!
```

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

If/Else in Karel

```
if (condition) {
    statement;
    statement;
    ...
} else {
    statement;
    statement;
    ...
}
```

Runs the first group of statements if ***condition*** is true; otherwise, runs the second group of statements.

While Loops in Karel

```
while (condition) {
    statement;
    statement;
    ...
}
```

Repeats the statements in the body until ***condition*** is no longer true.

Each time, Karel executes *all statements*, and **then** checks the condition.

Conditions in Karel

```
while(frontIsClear()) {
    body
}
```

```
if(beepersPresent()) {
    body
}
```

Conditions in Java

```
while(condition) {
    body
}
```

```
if(condition) {
    body
}
```

The condition should be a “boolean” which is either `true` or `false`

Conditions in Java

```
if (1 < 2) {
     println("1 is less than 2!");
}
```

```
int num = readInt("Enter a number: ");
if (num == 0) {
     println("That number is 0!");
} else {
     println("That number is not 0.");
}
```

Conditions in Java

```
int x = readInt("Enter a number: ");
while (x > 1) {
     x /= 2;
     println(x);
}
```

Output if the user enters 15:

```
7
3
1
```

Practice: Sentinel Loops

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for numbers until the user types -1 (sentinel), then output the sum of the numbers.

```
Type a number: 10
Type a number: 20
Type a number: 30
Type a number: -1
Sum is 60
```

Practice: Sentinel Loops

```
// fencepost problem!
// ask for number - post
// add number to sum - fence


int sum = 0;
int num = readInt("Enter a number: ");
while (num != SENTINEL) {
     sum += num;
     num = readInt("Enter a number: ");
}
println("Sum is " + sum);
```

Practice: Sentinel Loops

```
// Solution 2: Less repetition

int sum = 0;
while (true) {
     int num = readInt("Enter a number: ");
     if (num == SENTINEL) {
          break;     // immediately exits loop
     }
     sum += num;
}
println("Sum is " + sum);
```

Summary: Conditions

```
while(condition) {
    body
}
```

```
if(condition) {
    body
}
```

The condition should be a **boolean** which is either `true` or `false`

If/Else If/Else

```
if (condition1) {
    ...
} else if (condition2) {        // NEW
    ...
} else {
    ...
}
```

Runs the first group of statements if ***condition1*** is true; otherwise, runs the second group of statements if ***condition2*** is true; otherwise, runs the third group of statements.

You can have multiple else if clauses together.

If/Else If/Else

```
int num = readInt("Enter a number: ");
if (num > 5) {
     println("Your number is more than 5");
} else if (num > 2) {
     println("Your number is more than 2");
} else if (num > -1) {
     println("Your number is more than -1");
} else {
     println("Your number is negative");
}
```

Aside: Switch

- The **switch** statement is another way to easily do a limited form of cascaded if statements.

```
int day = readInt("Day of week (0-7)");
switch (day) {
    case 0:
            println("Sunday");
            break;
    case 6:
            println("Saturday");
            break;
    default:
            println("Weekday");
            break;
}
```

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

For Loops in Karel

```
for (int i = 0; i < max; i++) {
    statement;
    statement;
    ...
}
```

Repeats the statements in the body ***max*** times.

For Loops in Java

```
for (init; test; step) {
    statement;
    statement;
    ...
}
```

For Loops in Java

This code is run once, just before the for loop starts

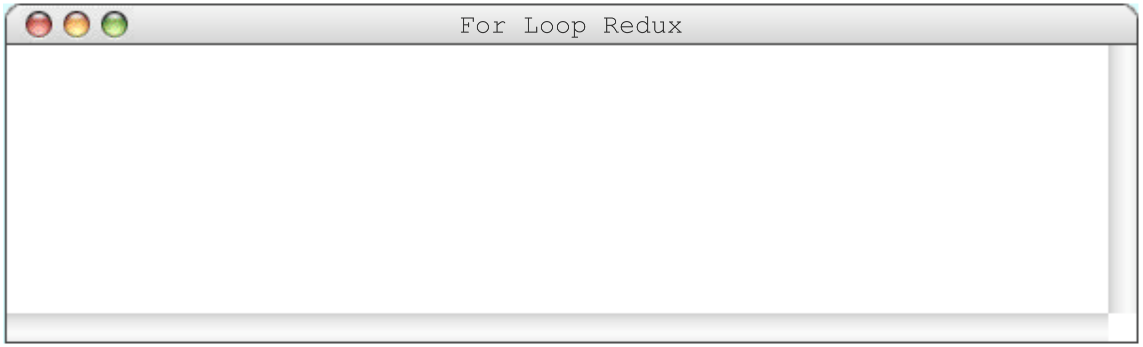
Execute the loop if this condition passes

This code is run each time the code gets to the end of the 'body'

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

For Loops in Java

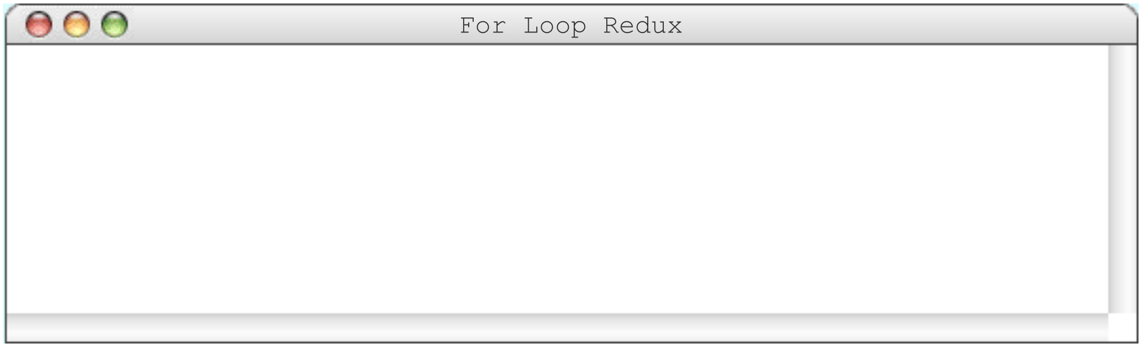
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



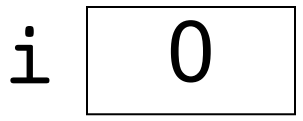
For Loops in Java

```
i  0
```

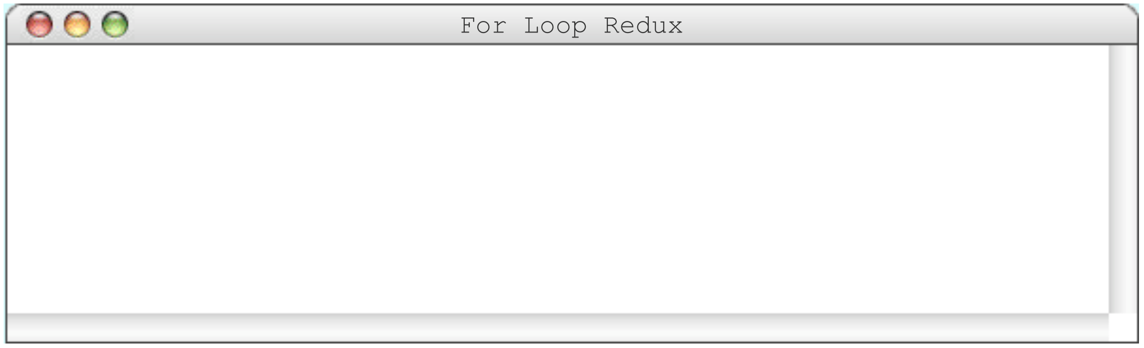
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



For Loops in Java



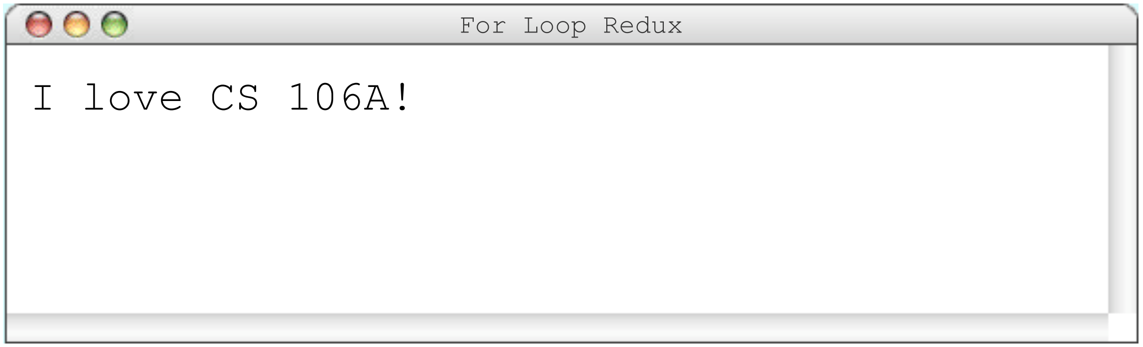
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



For Loops in Java

```
i  0
```

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

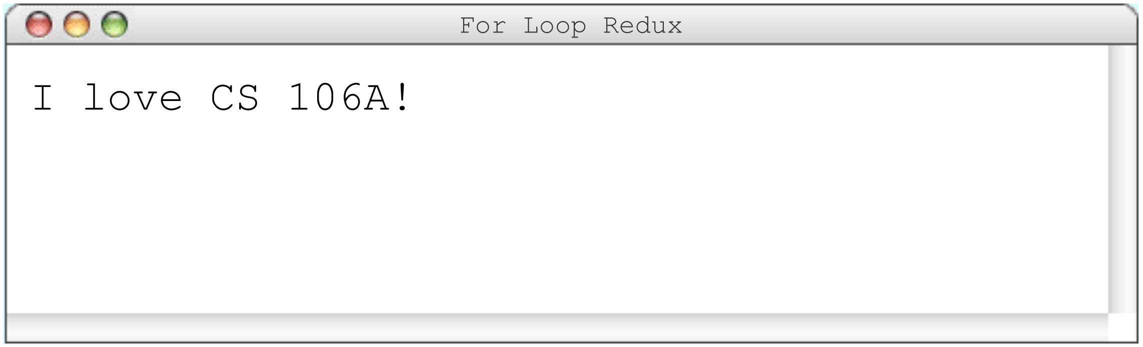


```
I love CS 106A!
```

For Loops in Java

i $\boxed{0}$

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

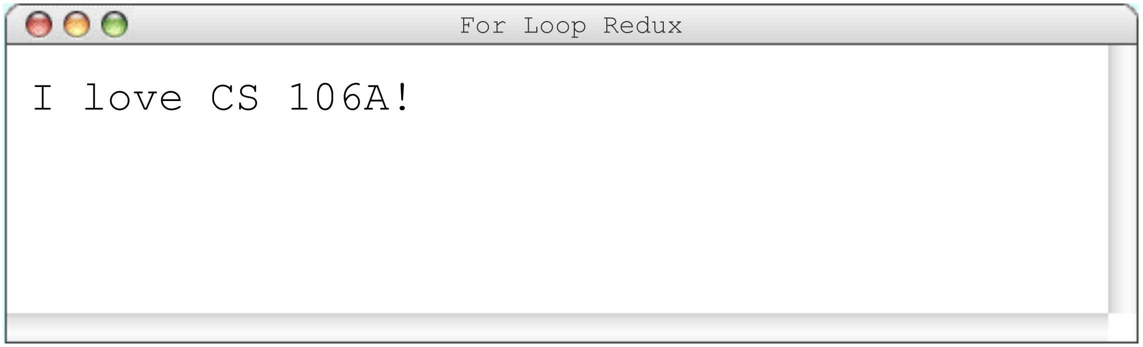

For Loop Redux

```
I love CS 106A!
```

For Loops in Java

i $\boxed{1}$

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

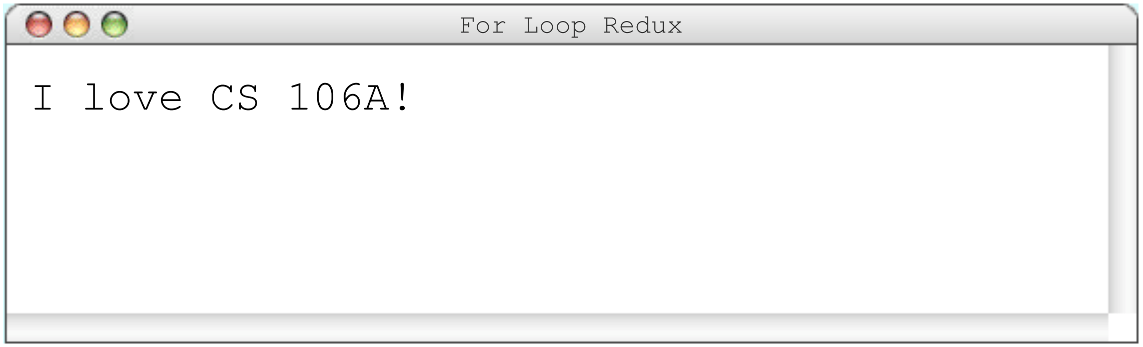


```
I love CS 106A!
```

For Loops in Java

```
i  1
```

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

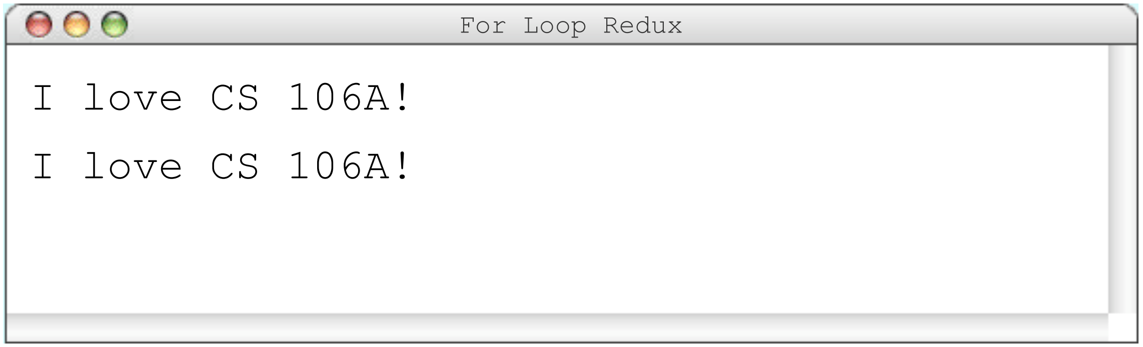


```
I love CS 106A!
```

For Loops in Java

$i \ \boxed{1}$

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

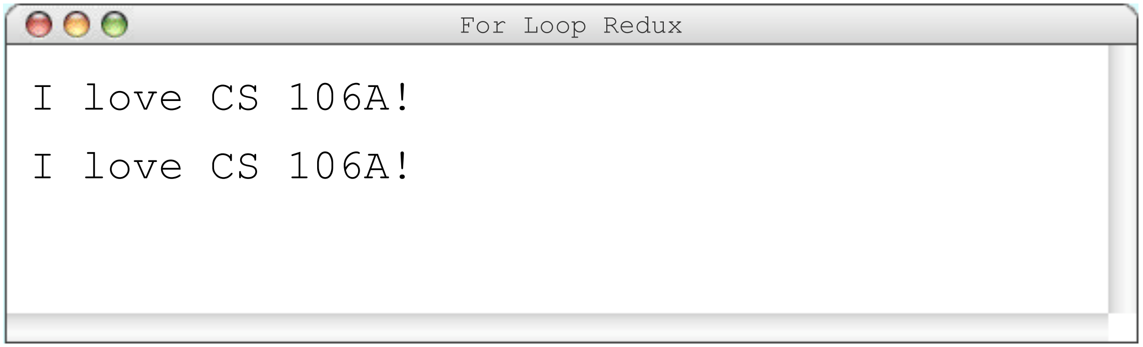


```
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  2
```

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

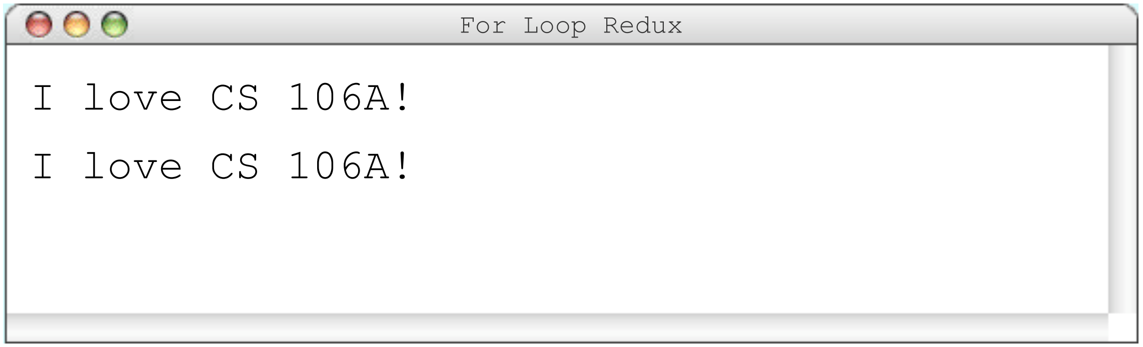


```
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  2
```

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

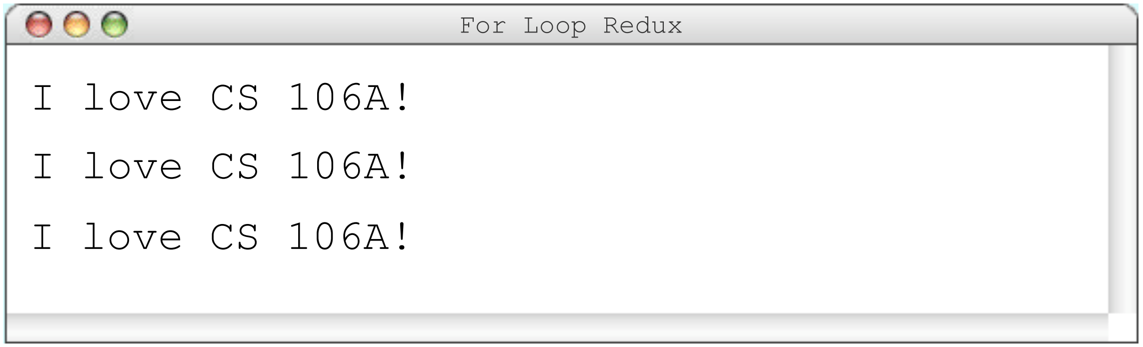

For Loop Redux

```
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  2
```

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

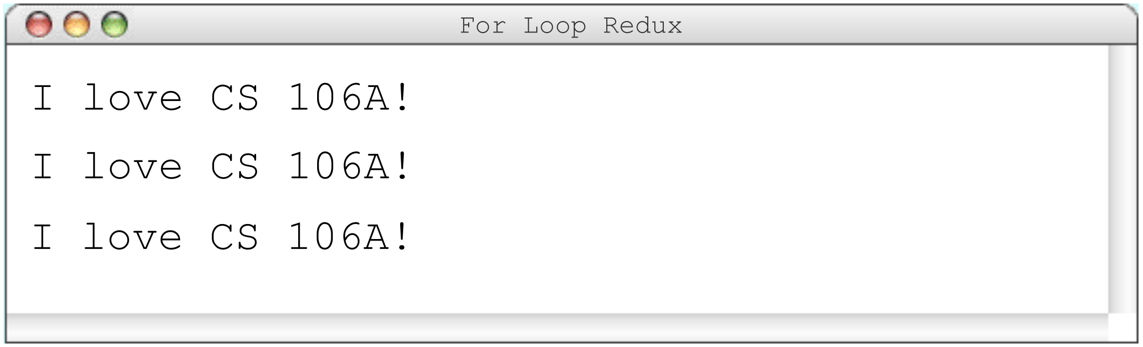


```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  3
```

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```

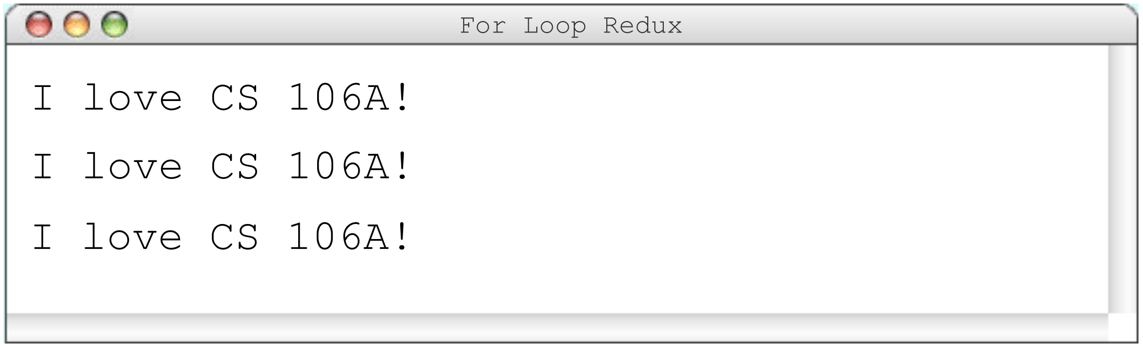

For Loop Redux

```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  3
```

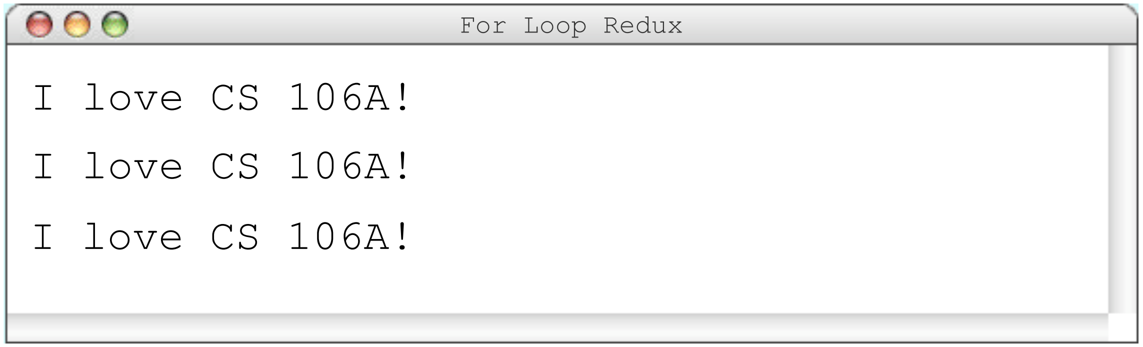
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

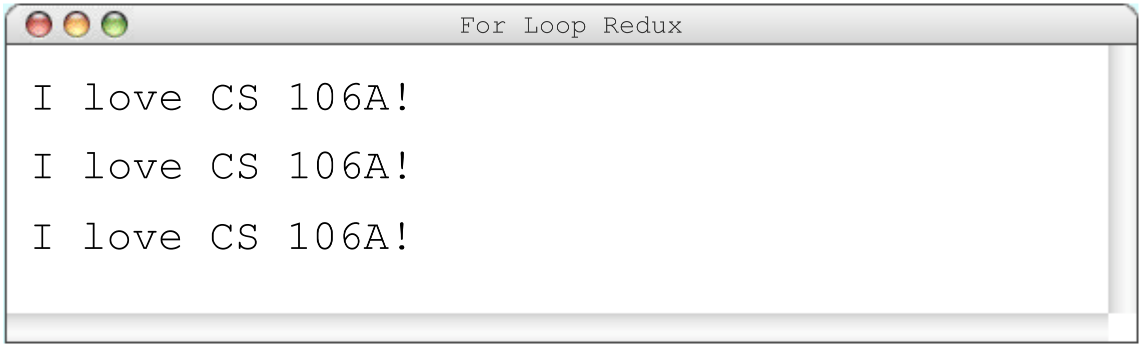
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```java
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

Using the For Loop Variable

```
for(int i = 0; i < 5; i++) {
    println(i);
}
```

```
0
1
2
3
4
```

Using the For Loop Variable

```
// Launch countdown
for(int i = 10; i >= 1; i--) {
    println(i);
}
println("Blast off!");
```

Output:

```
10
9
8
…
Blast off!
```

Using the For Loop Variable

```
int sum = 0;
for(int i = 1; i <= 5; i += 2) {
    sum += i;
}
println("The sum is " + sum);
```

Output:

```
The sum is 9
```

`for` versus `while`

```
for (init ; test ; step) {
        statements
}
```

- **`for`** loop used for *definite* iteration
- Generally, we know how many times we want to iterate

```
init
while ( test ) {
        statements
        step
}
```

- **`while`** loop used for *indefinite* iteration
- Generally, don’t know how many times to iterate beforehand

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

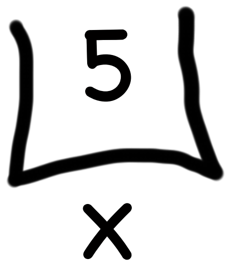
A Variable love story

By Chris Piech

Once upon a time…

...x was looking for love!

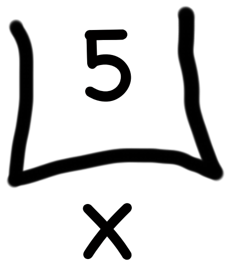
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



...x was looking for love!

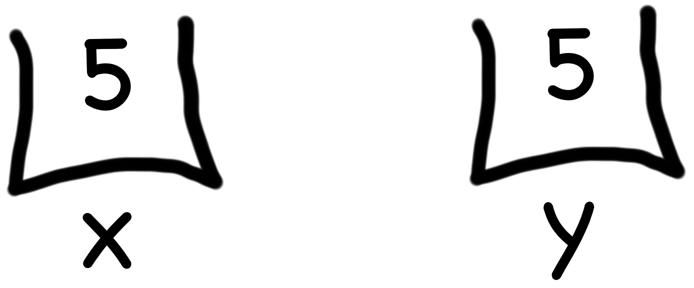
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

x was definitely looking for love



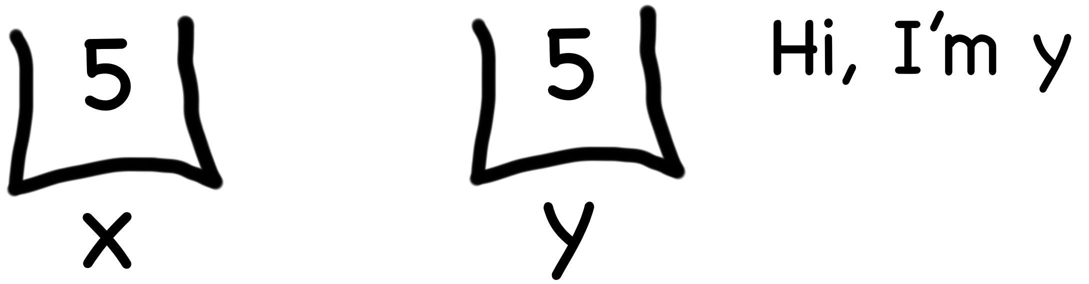
And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



And met y.

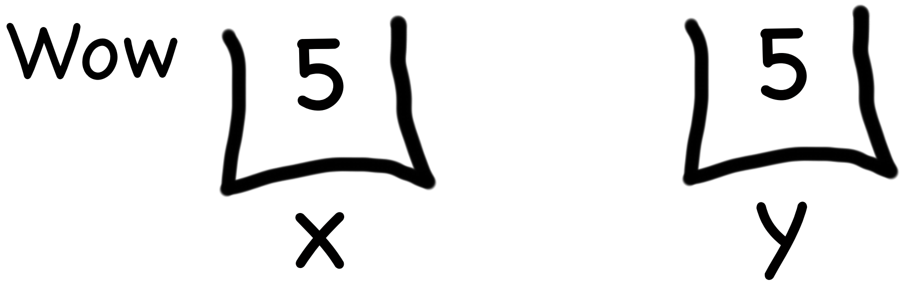
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



“Wow!”

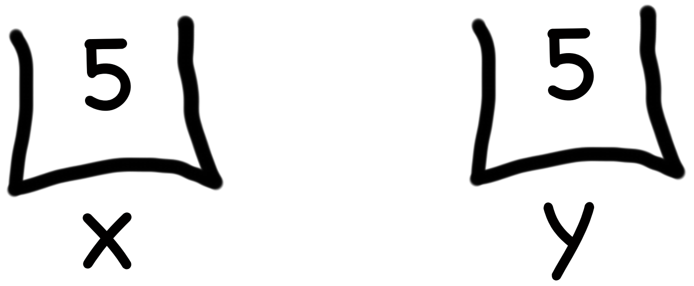
And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



And met y.

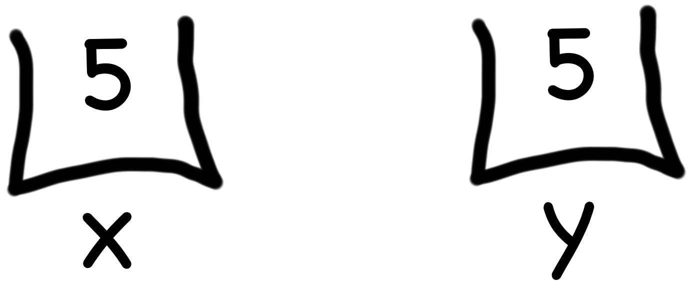
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



We have so much in common

And met y.

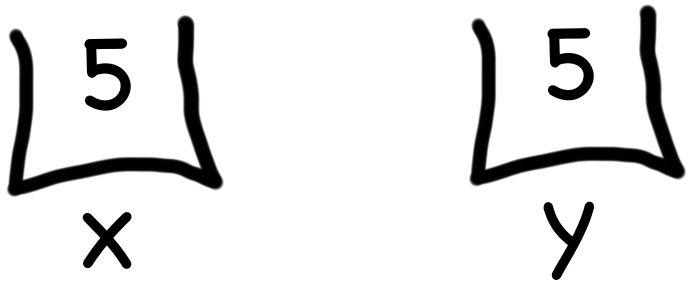
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



We both have value 5!

And met y.

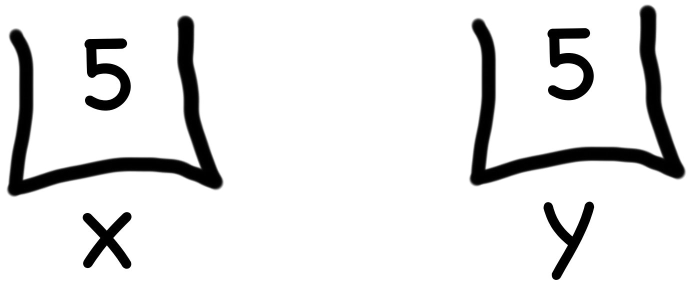
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



Maybe sometime
we can…

And met y.

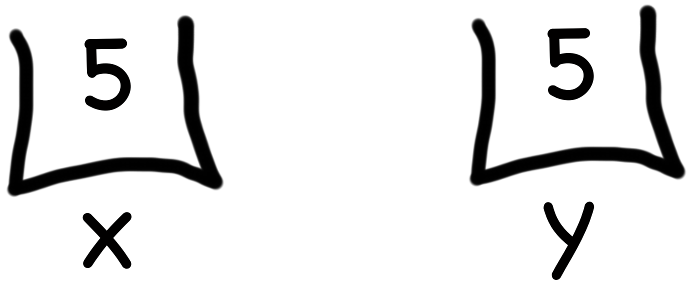
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



println together?

And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

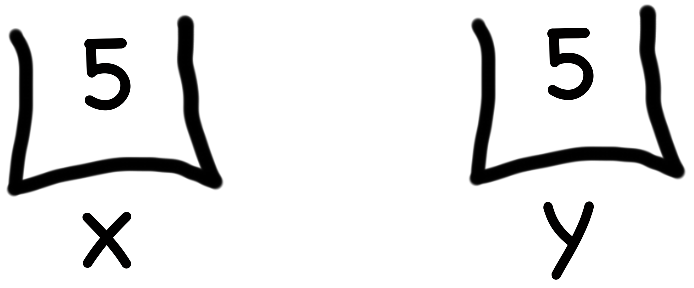


It was a beautiful match…

…but then tragedy struck.

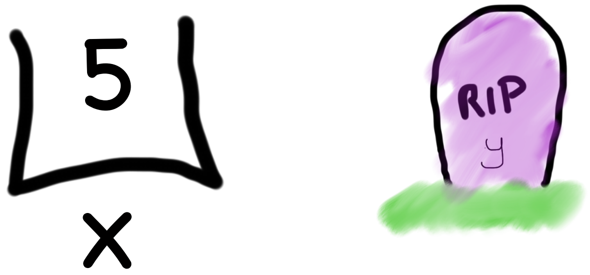
Tragedy Strikes

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



Tragedy Strikes

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

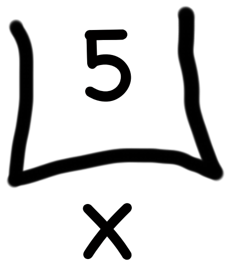


Noooooooooooooooo!

You see…
when a program exits a code block,
all variables declared inside that block go away!

Since y is inside the if-block...

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



...it goes away here...

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

...and doesn't exist here.

```
int x = 5;
if(lookingForLove()) {
   int y = 5;
}
println(x + y);
```

Error. Undefined variable y.

The End

Sad times ☹

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Variable Scope

Variables have a lifetime (called scope):

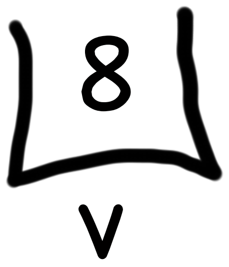
```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Comes to life here

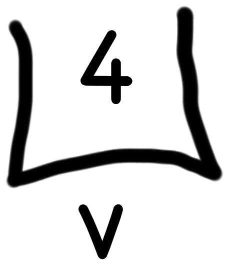


Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

This is the **inner most** code block in which it was declared....

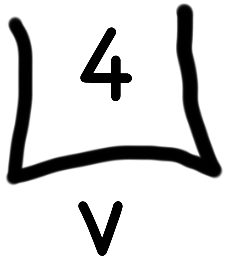


Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Still alive here...

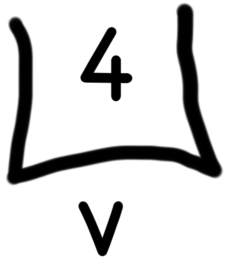


Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

It goes away here (at the end of its code block)



Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

It goes away here (at the end of its code block)

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    ... some code
    if(condition){
        int w = 4;
        ... some code
    }
    ... some other code
}
```

This is the scope of **w**

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    … some code
    if(condition){
        int w = 4;
        … some code
    }
    … some other code
}
```

w is created here

w goes away here (at the end of its code block)

Variable Scope

```
public void run() {
      int x = readInt("Number: ");
      if (x < 2) {
            int y = 4;
      }

      // ERROR!  "Undefined variable y"
      println("Y has the value " + y);
}
```

A Variable love story

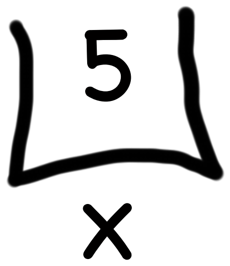
Chapter 2

By Chris

The programmer fixed the bug

...x was looking for love!

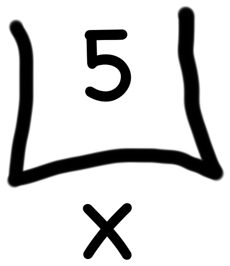
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```



...x was looking for love!

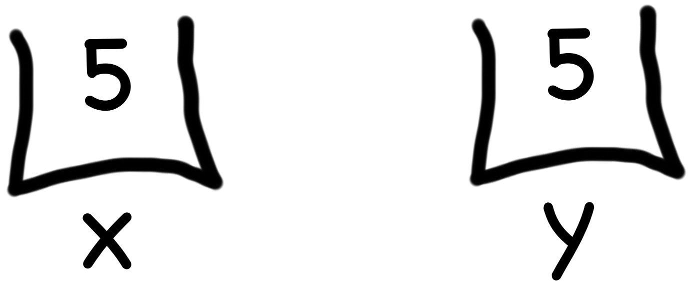
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```

x was definitely looking for love



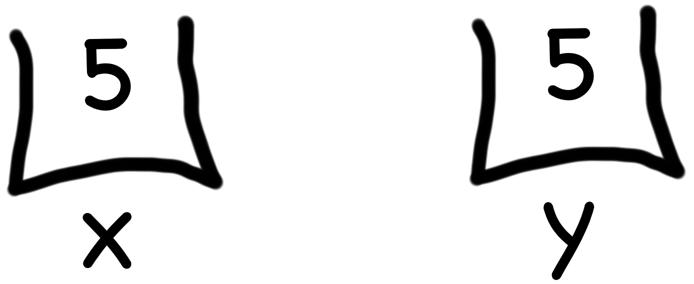
And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```



Since they were both "in scope"...

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```



…they lived happily ever after.
The end.

Variable Scope

- The **scope** of a variable refers to the section of code where a variable can be accessed.
- **Scope starts** where the variable is declared.
- **Scope ends** at the termination of the statement block in which the variable was declared.

- A **statement block** is a chunk of code between { } brackets

Variable Scope

You *cannot* have two variables with the same name in the *same scope*.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;                  // ERROR
    print("/");
}
```

Variable Scope

You *can* have two variables with the same name in *different scopes*.

```
public void run() {
  for (int i = 0; i < 5; i++) {
      int y = 2;
      ...
  }

  for (int i = 10; i >= 0; i--) {
      int y = 3;      // ok
  }
```

Revisiting Sentinel Loops

```
// sum must be outside the while loop!
// Otherwise it will be redeclared many times.
int sum = 0;
int num = readInt("Enter a number: ");
while (num != -1) {
     sum += num;
     num = readInt("Enter a number: ");
}
println("Sum is " + sum);
```

Plan For Today

- Announcements
- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard (see handout 12)

Recap

- Recap: Expressions and Booleans
- Aside: Strings
- Revisiting Control Flow
  - If and While
  - For
  - Scope
- Example: Checkerboard

Next time: Methods in Java