

CS 106A, Lecture 5

Booleans, Control Flow and Scope

suggested reading:

Java Ch. 3.4-4.6

Plan For Today

- Announcements
- Recap: Java, Variables and Expressions
- Aside: Shorthand Operators + Constants
- Revisiting Control Flow
 - If and While
 - For
- Scope

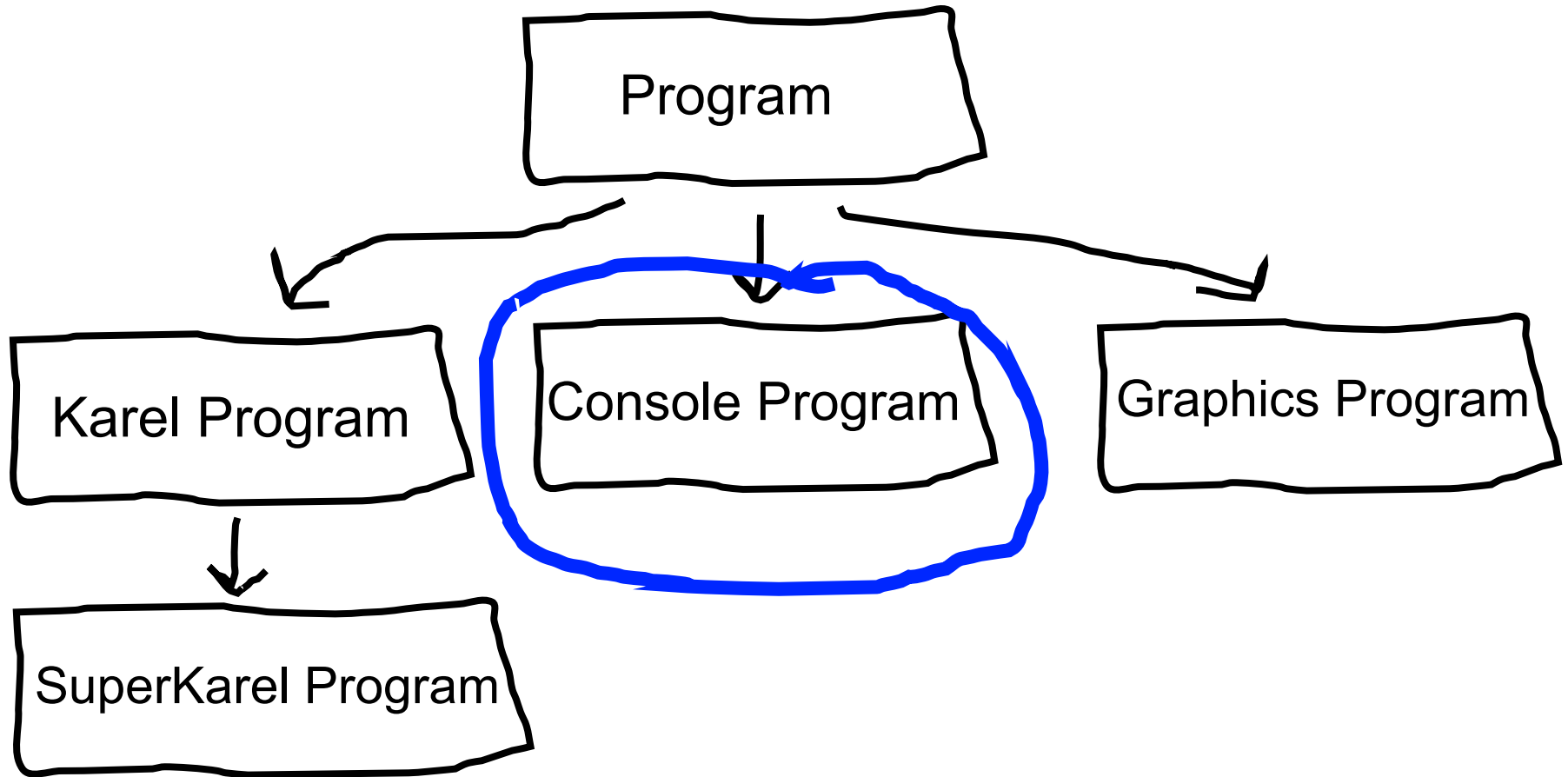
Plan For Today

- Announcements
- Recap: Java, Variables and Expressions
- Aside: Shorthand Operators + Constants
- Revisiting Control Flow
 - If and While
 - For
- Scope

Plan For Today

- Announcements
- **Recap: Java, Variables and Expressions**
- Aside: Shorthand Operators + Constants
- Revisiting Control Flow
 - If and While
 - For
- Scope

Java



Console Programs

```
import acm.program.*;

public class Name extends ConsoleProgram {
    public void run() {
        statements;
    }
}
```

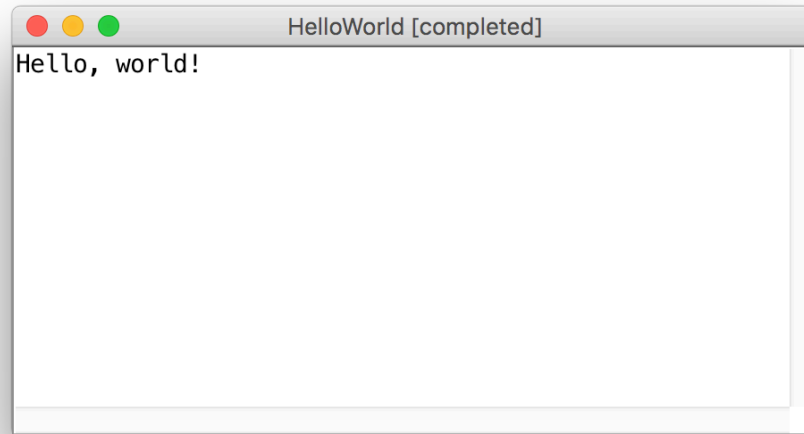
- Unlike Karel, many programs produce their behavior as text.
- **console:** Text box into which the behavior is displayed.
 - *output:* Messages displayed by the program.
 - *input:* Data read by the program that the user types.

println

- A statement that prints a line of output on the console, and goes to the next line.
 - pronounced "print-linn"
- Two ways to use println :
 - `println("text");`
 - Prints the given message as output, and goes to the next line.
 - A message is called a *string*; it starts/ends with a " quote character.
 - The quotes do not appear in the output.
 - A string may not contain a " character.
 - `println();`
 - Prints a blank line of output.

print

```
public class HelloWorld extends ConsoleProgram {  
    public void run() {  
        print("Hello, ");  
        print("world!");  
    }  
}
```



Same as `println`, but does not go to the next line.

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

Type Interactions

`int` and `int` results in an `int`

`double` and `double` results in a `double`

`int` and `double` results in a `double`

`String` and `int` results in a `String`

etc.

* The general rule is: operations always return the most expressive type

Integer division

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

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

$$\begin{array}{r} 3 \\ 4 \overline{) 14} \\ \underline{12} \\ 2 \end{array}$$

$$\begin{array}{r} 4 \\ 10 \overline{) 45} \\ \underline{40} \\ 5 \end{array}$$

$$\begin{array}{r} 52 \\ 27 \overline{) 1425} \\ \underline{135} \\ 75 \\ \underline{54} \\ 21 \end{array}$$

- More examples:

– 32 / 5 is 6

– 84 / 10 is 8

– 156 / 100 is 1

– Dividing by 0 causes an error when your program runs.

Practice

- $1 / 2$
- $1.0 / 2$
- $1 + 2 / 3$
- `"abc" + (4 + 2)`
- `"abc" + 4 + 2`

Making a new Variable

type



name



```
int myVariable;
```

Variable Types

int – an integer number

double – a decimal number

Assignment

Existing variable name



value



```
myVariable = 2;
```


Assignment

- **assignment:** Stores a value into a variable.
 - The value can be an expression; the variable stores its result.

- Syntax:

name = expression;

```
int zipcode;  
zipcode = 90210;
```

zipcode

90210

```
double myGPA;  
myGPA = 1.0 + 2.25;
```

myGPA

3.25

Declare / initialize

- A variable can be declared/initialized in one statement.
 - This is probably the most commonly used declaration syntax.

- Syntax:

type name = expression;

```
double tempF = 98.6;
```

tempF

98.6

```
int x = (12 / 2) + 3;
```

x

9

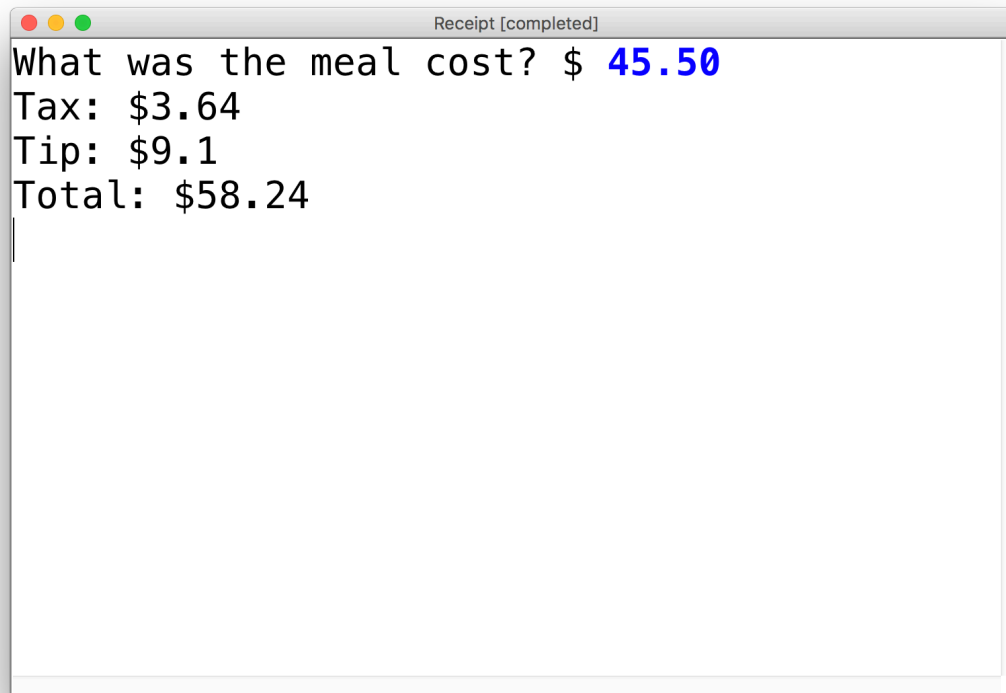
Using Variables

```
// Asks the user for an integer by  
// displaying the given message  
// and stores it in the variable 'a'  
int a = readInt(message);
```

```
// Asks the user for a double by  
// displaying the given message and  
// stores it in the variable 'b'  
double b = readDouble(message);
```

Practice: Receipt Program

- We wrote a ConsoleProgram called *Receipt* that calculates the tax, tip and total bill for us at a restaurant.
- The program asks the user for the subtotal, and then calculate and print out the tax, tip and total.



```
Receipt [completed]
What was the meal cost? $ 45.50
Tax: $3.64
Tip: $9.1
Total: $58.24
```

Plan For Today

- Announcements
- Recap: Variables and Expressions
- **Aside: Shorthand Operators + Constants**
- Revisiting Control Flow
 - If and While
 - For
- Scope

Shorthand Operators

Shorthand

variable += *value*;

variable -= *value*;

variable *= *value*;

variable /= *value*;

variable %= *value*;

variable++;

variable--;

x += 3;

number *= 2;

x++;

Equivalent longer version

variable = *variable* + *value*;

variable = *variable* - *value*;

variable = *variable* * *value*;

variable = *variable* / *value*;

variable = *variable* % *value*;

variable = *variable* + 1;

variable = *variable* - 1;

// x = x + 3;

// number = number * 2;

// x = x + 1;

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.

- 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;
```

Receipt Program - Before

```
public class Receipt extends ConsoleProgram {  
    public void run() {  
        double subtotal = readDouble("Meal cost? $");  
        double tax = subtotal * 0.08;  
        double tip = subtotal * 0.20;  
        double total = subtotal + tax + tip;  
  
        println("Tax : $" + tax);  
        println("Tip: $" + tip);  
        println("Total: $" + total);  
    }  
}
```


Receipt Program – After

```
public class Receipt extends ConsoleProgram {  
    private static final double TAX_RATE = 0.08;  
    private static final double TIP_RATE = 0.2;  
  
    public void run() {  
        double subtotal = readDouble("Meal cost? $");  
        double tax = subtotal * TAX_RATE;  
        double tip = subtotal * TIP_RATE;  
        double total = subtotal + tax + tip;  
  
        println("Tax : $" + tax);  
        println("Tip: $" + tip);  
        println("Total: $" + total);  
    }  
}
```

Plan For Today

- Announcements
- Recap: Variables and Expressions
- Aside: Shorthand Operators + Constants
- **Revisiting Control Flow**
 - If and While
 - For
- Scope

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**

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
<code>==</code>	equals	<code>1 + 1 == 2</code>	true
<code>!=</code>	does not equal	<code>3.2 != 2.5</code>	true
<code><</code>	less than	<code>10 < 5</code>	false
<code>></code>	greater than	<code>10 > 5</code>	true
<code><=</code>	less than or equal to	<code>126 <= 100</code>	false
<code>>=</code>	greater than or equal to	<code>5.0 >= 5.0</code>	true

* All have equal precedence

Relational Operators

```
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.");  
}
```

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, then output the sum of the numbers.
 - In this case, -1 is the sentinel value.

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 <= -1) {  
    sum += num;  
    num = readInt("Enter a number: ");  
}  
println("Sum is " + sum);
```

Practice: Sentinel Loops

// Solution #2 (ok, but #1 is better)

```
int sum = 0;
while (true) {
    int num = readInt("Enter a number: ");
    if (num == -1) {
        break;        // immediately exits loop
    }
    sum += num;
}
println("Sum is " + sum);
```

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

// 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 isFamilyVisiting = true;
```

```
boolean isRaining = false;
```

Boolean Variables

```
// 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 isFamilyVisiting = true;
```

```
boolean isRaining = false;
```

```
// Ask the user a true/false (yes/no) question
```

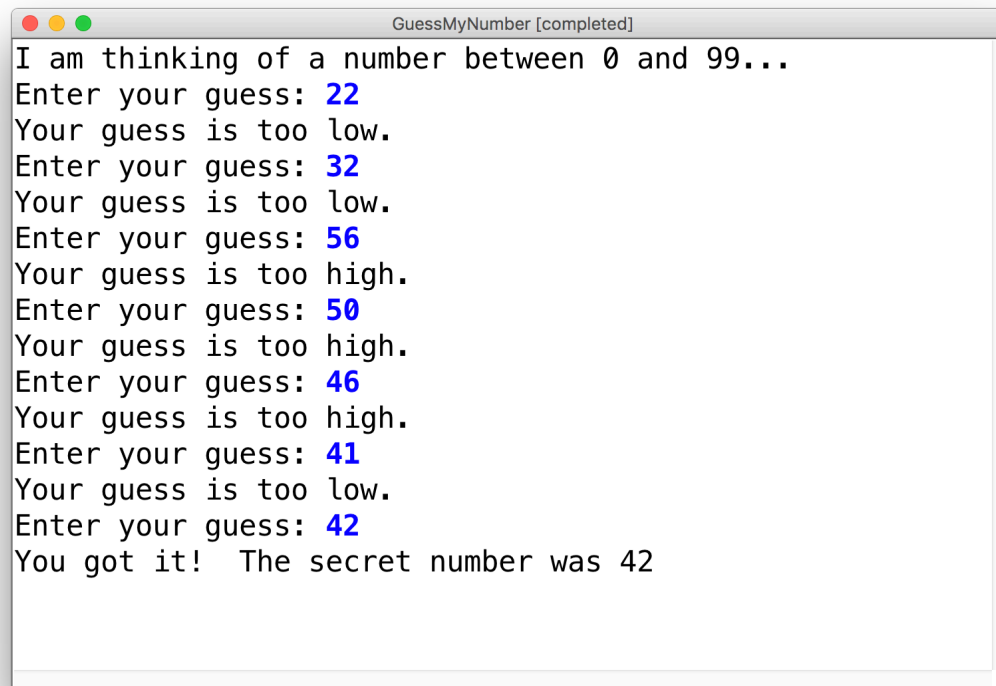
```
boolean playAgain = readBoolean("Play again?");
```

```
if (playAgain) {
```

```
...
```

Practice: GuessMyNumber

- Let's write a program called *GuessMyNumber* that prompts the user for a number until they guess our secret number.
- If a guess is incorrect, the program should provide a hint; specifically, whether the guess is too high or too low.



```
GuessMyNumber [completed]
I am thinking of a number between 0 and 99...
Enter your guess: 22
Your guess is too low.
Enter your guess: 32
Your guess is too low.
Enter your guess: 56
Your guess is too high.
Enter your guess: 50
Your guess is too high.
Enter your guess: 46
Your guess is too high.
Enter your guess: 41
Your guess is too low.
Enter your guess: 42
You got it! The secret number was 42
```

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 > 0) {  
    println("Your number is positive");  
} else if (num < 0) {  
    println("Your number is negative");  
} else {  
    println("Your number is 0");  
}
```

Plan For Today

- Announcements
- Recap: Variables and Expressions
- Aside: Shorthand Operators + Constants
- Revisiting Control Flow
 - If and While
 - **For**
- Scope

For Loops in Karel

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

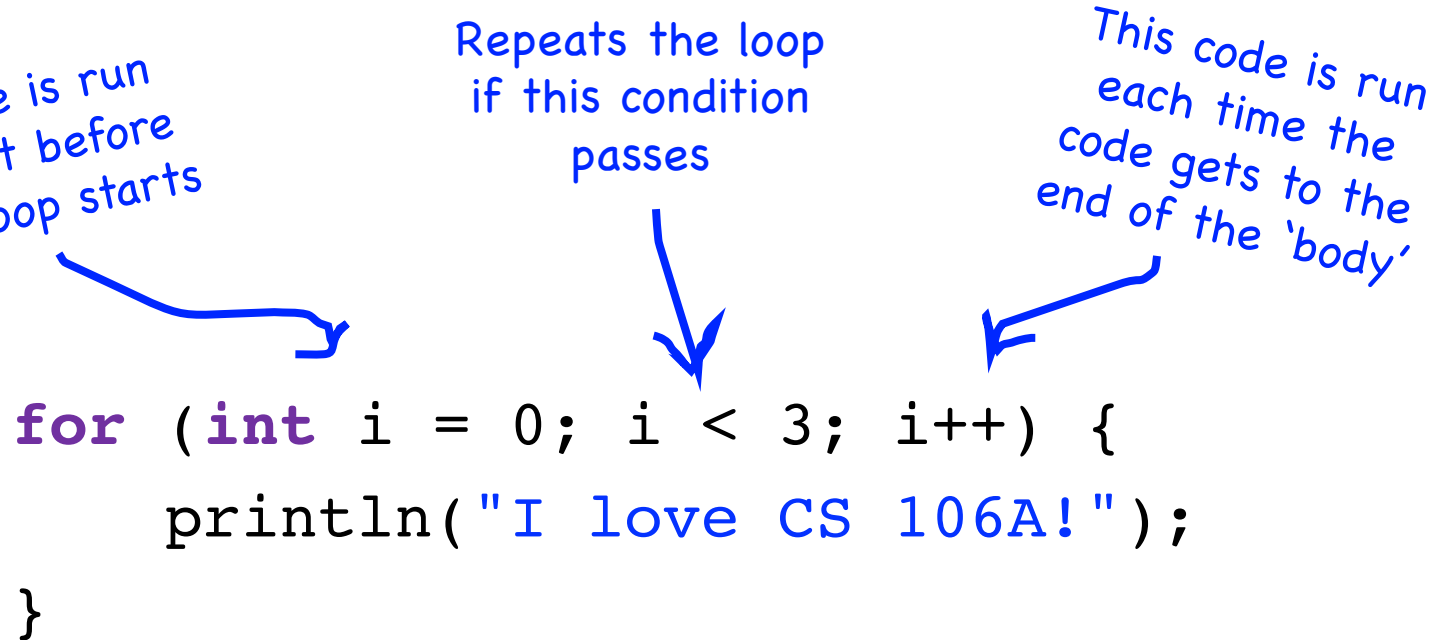
Repeats the statements in the body *max* times.

For Loops in Java

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

*Repeats the loop
if this condition
passes*

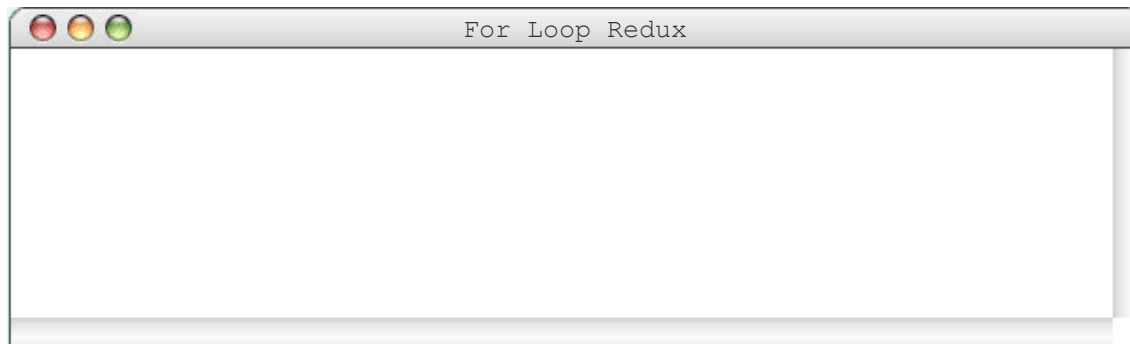
*This code is run
each time the
code gets to the
end of the 'body'*



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

For Loops in Java

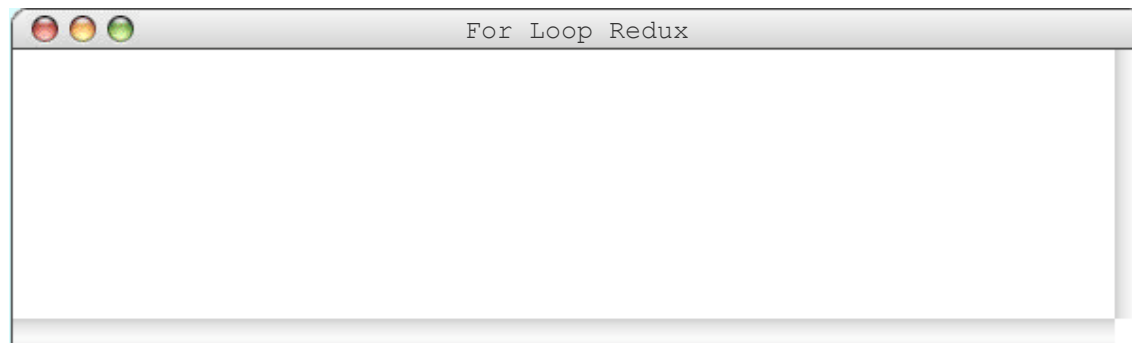
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 0

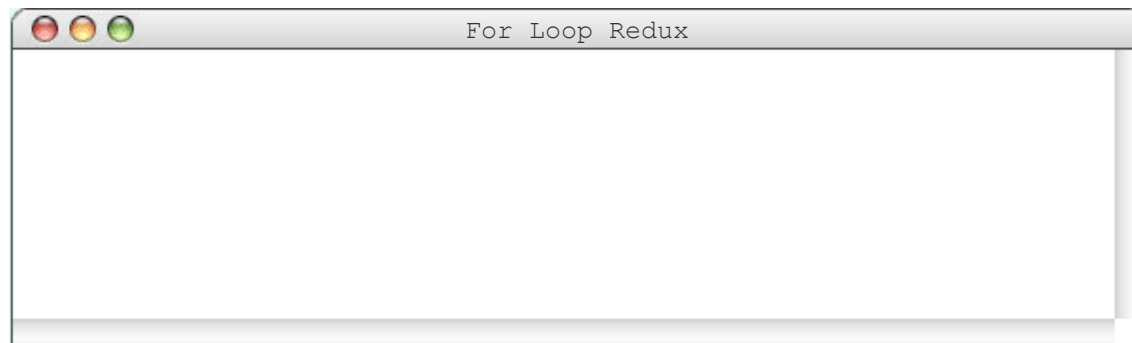
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 0

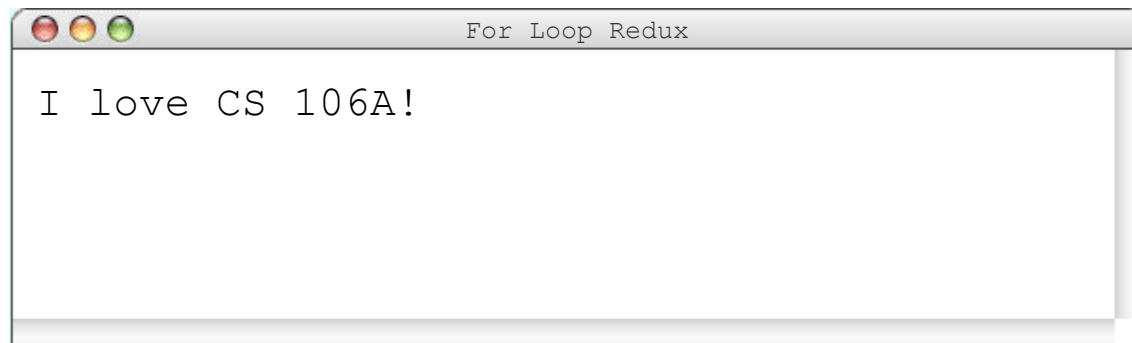
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 0

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

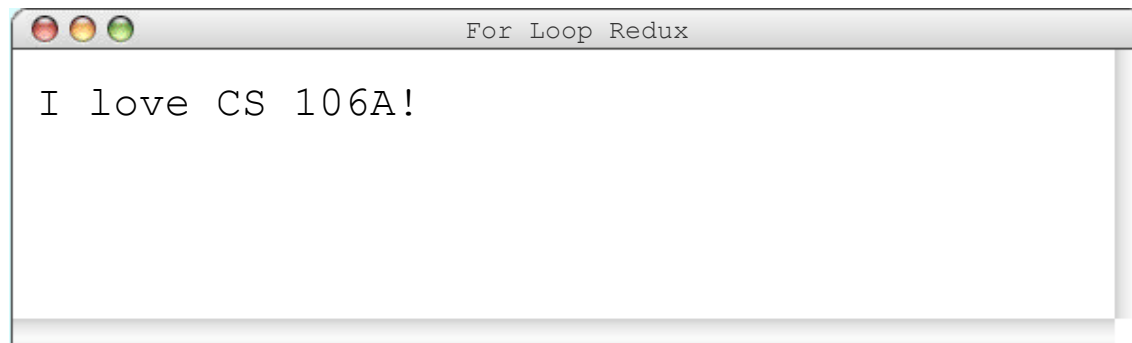


For Loops in Java

i 0

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

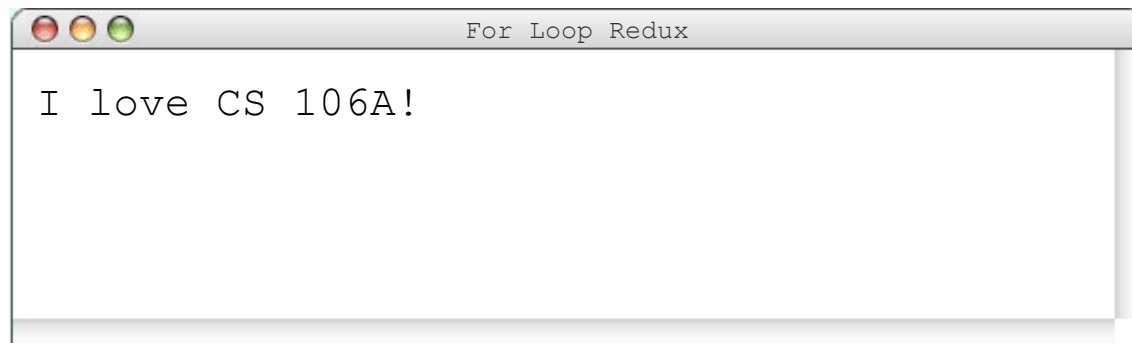
```
}
```



For Loops in Java

i 1

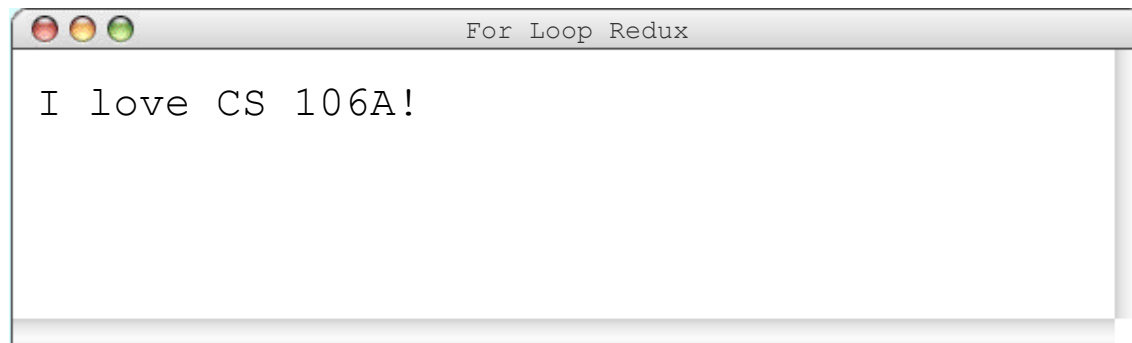
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 1

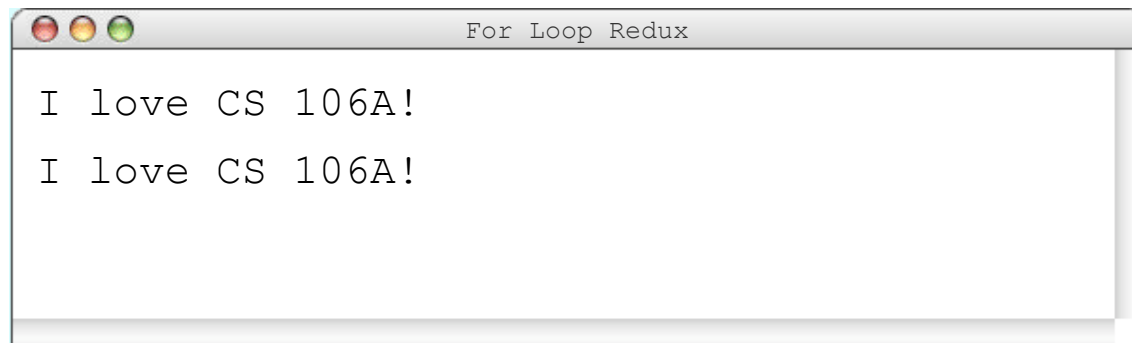
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 1

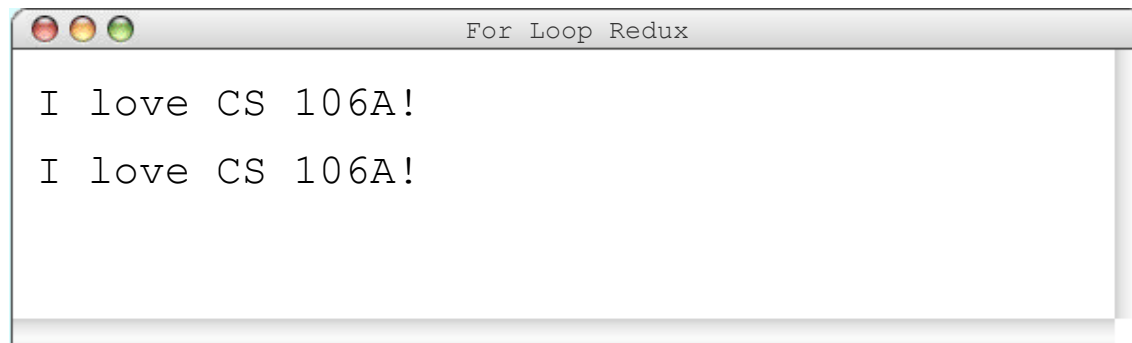
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 2

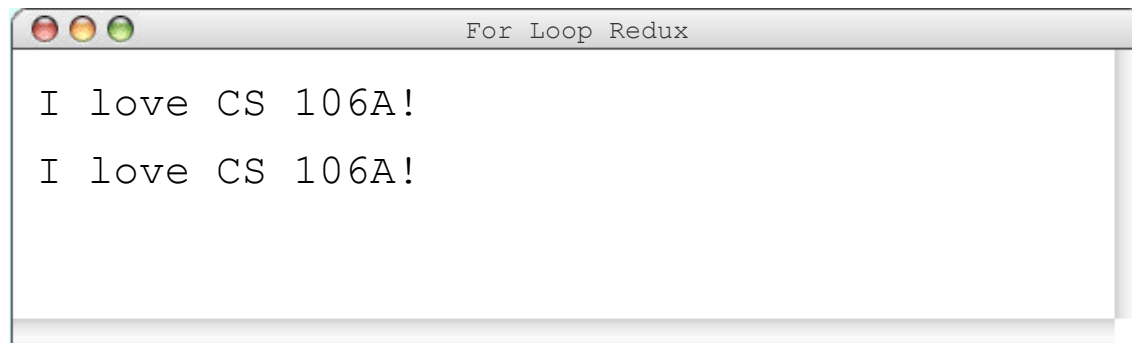
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 2

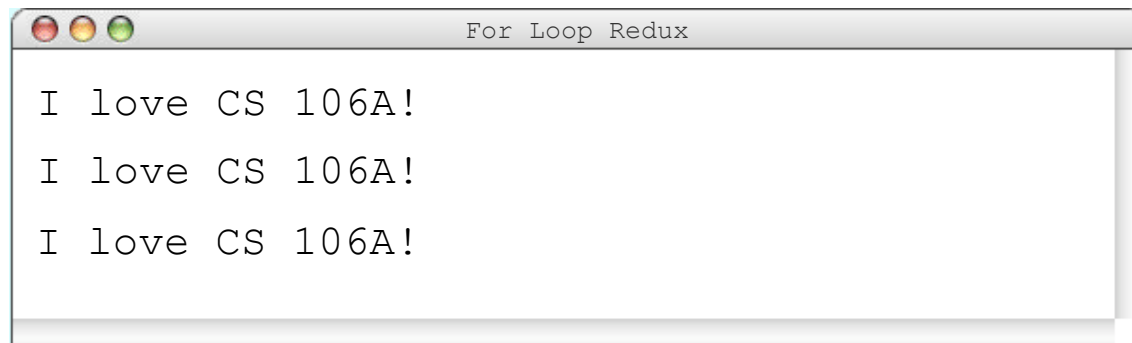
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 2

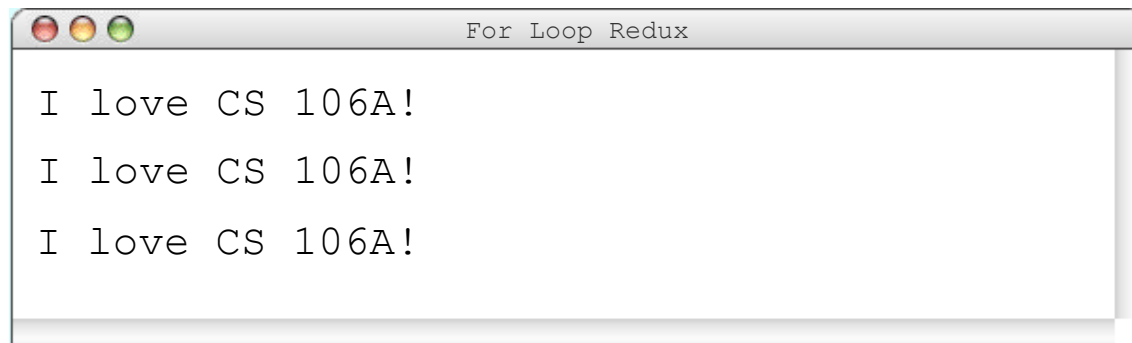
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

i 3

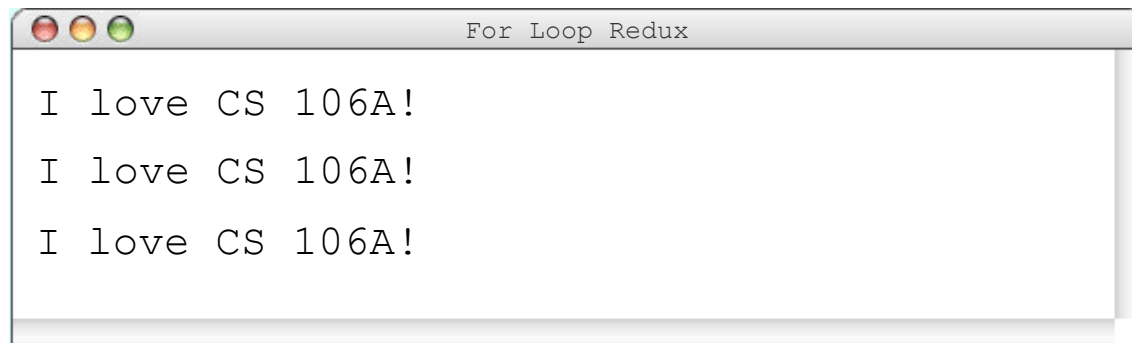
```
for (int i = 0; i < 3; i++) {  
    println("I love CS 106A!");  
}
```



For Loops in Java

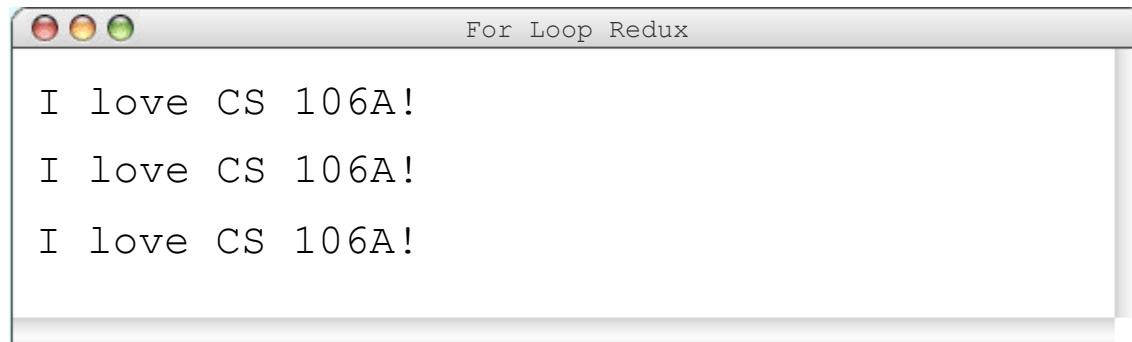
i 3

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



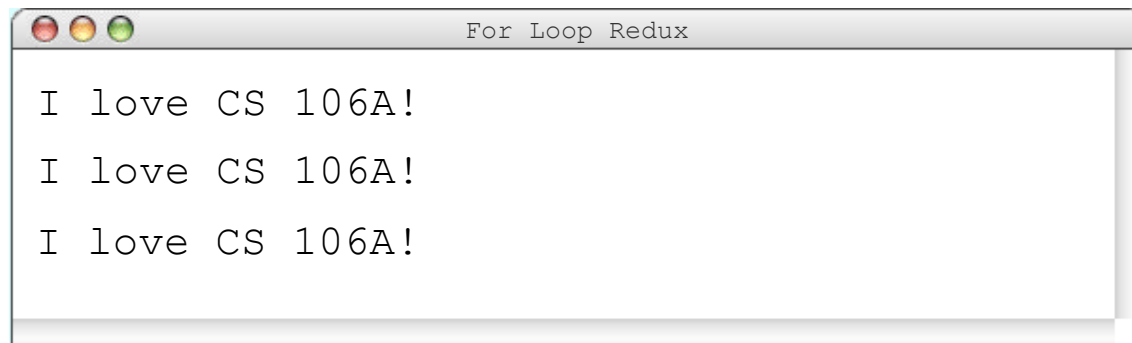
For Loops in Java

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



For Loops in Java

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



Using the For Loop Variable

```
// prints the first 100 even numbers
for(int i = 0; i < 100; i++) {
    println(i * 2);
}
```

Using the For Loop Variable

```
// Launch countdown
for(int i = 10; i >= 1; i--) {
    println(i * 2);
}
println("Blast off!");
```

Output:

10

9

8

...

Blast off!

Using the For Loop Variable

```
// Adds up the first 100 numbers
int sum = 0;
for(int i = 0; i < 100; i++) {
    sum += i;
}
println("The sum is " + sum);
```

Nested loops

- **nested loop:** A loop placed inside another loop.

```
for (int i = 1; i <= 5; i++) {  
    for (int j = 1; j <= 10; j++) {  
        print("*");  
    }  
    println();    // to end the line  
}
```

- Output:

```
*****  
*****  
*****  
*****  
*****
```

- The outer loop repeats 5 times; the inner one 10 times.

Nested loop question

- **Q:** What output is produced by the following code?

```
for (int i = 1; i <= 5; i++) {  
    for (int j = 1; j <= i; j++) {  
        print("*");  
    }  
    println();  
}
```

- | | | | | |
|-----------|-----------|-----------|-----------|-----------|
| A. | B. | C. | D. | E. |
| ***** | ***** | * | 1 | 12345 |
| ***** | **** | ** | 22 | |
| ***** | *** | *** | 333 | |
| ***** | ** | **** | 4444 | |
| ***** | * | ***** | 55555 | |

(How would you modify the code to produce each output above?)

Plan For Today

- Announcements
- Recap: Variables and Expressions
- Aside: Shorthand Operators + Constants
- Revisiting Control Flow
 - If and While
 - For
- **Scope**

A Variable love story

By Chris Piech

Once upon a time...

...x was looking for love!

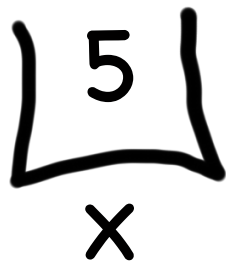
```
int x = 5;
```

```
if(lookingForLove()) {
```

```
    int y = 5;
```

```
}
```

```
println(x + y);
```



A hand-drawn diagram consisting of a curly brace that groups the number 5. Below the brace is the letter 'x', indicating that the variable x holds the value 5.

...x was looking for love!

```
int x = 5;
```

```
if(lookingForLove()) {
```

```
    int y = 5;
```

```
}
```

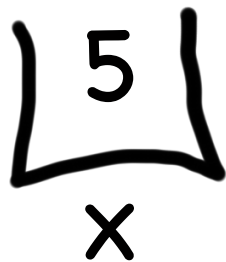
```
println(x + y);
```

x was definitely
looking for love

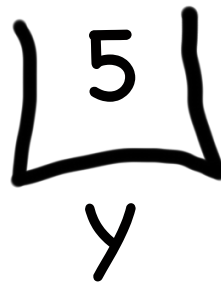
A hand-drawn diagram consisting of a large, hand-drawn curly brace. Inside the top part of the brace is the number '5'. Below the bottom of the brace is the variable name 'x'. This diagram visually represents the state of the variable x, which holds the value 5.

And met y.

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



A hand-drawn diagram showing the number 5 inside a bracket shape, with the letter x written below it.



A hand-drawn diagram showing the number 5 inside a bracket shape, with the letter y written below it.

And met y.

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

5
x

5
y

Hi, I'm y

“Wow!”

And met y.

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

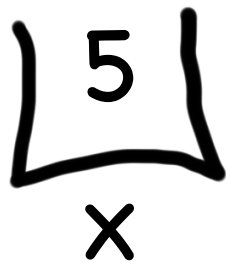
Wow

5
x

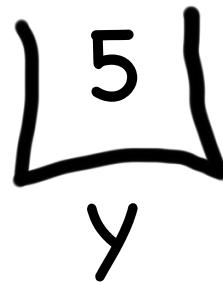
5
y

And met y.

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



A hand-drawn diagram consisting of a large, slightly irregular bracket shape. Inside the bracket is the number 5. Below the bracket is the letter x.

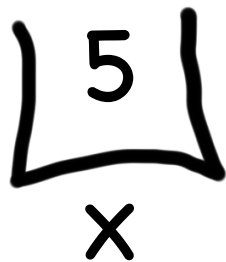


A hand-drawn diagram consisting of a large, slightly irregular bracket shape. Inside the bracket is the number 5. Below the bracket is the letter y.

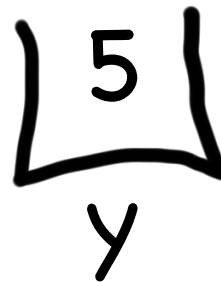
We have so much
in common

And met y.

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



A hand-drawn diagram showing a variable `x` containing the value 5. The number 5 is enclosed in a hand-drawn box, and the letter `x` is written below it.

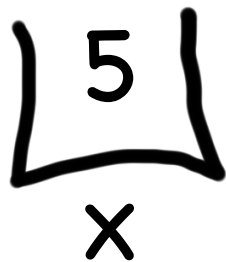


A hand-drawn diagram showing a variable `y` containing the value 5. The number 5 is enclosed in a hand-drawn box, and the letter `y` is written below it.

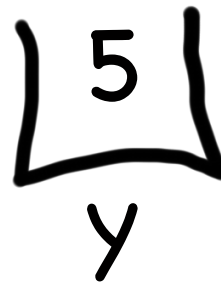
We both have
value 5!

And met y.

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



A hand-drawn diagram consisting of a box with a curved bottom, containing the number 5. Below the box is the letter x.



A hand-drawn diagram consisting of a box with a curved bottom, containing the number 5. Below the box is the letter y.

Maybe sometime
we can...

And met y.

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

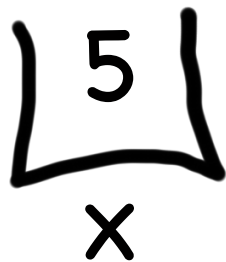
5
x

5
y

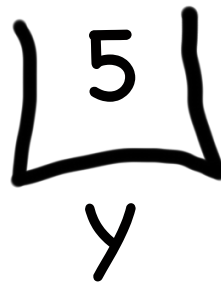
println together?

And met y.

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



A hand-drawn diagram showing the number 5 inside a bracket shape. Below the bracket is the letter x.



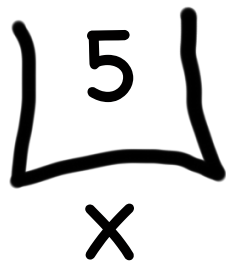
A hand-drawn diagram showing the number 5 inside a bracket shape. Below the bracket is the letter y.

It was a beautiful match...

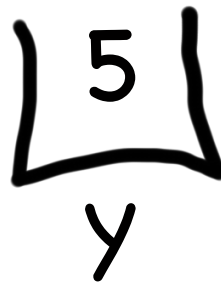
...but then tragedy struck.

Tragedy Strikes

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



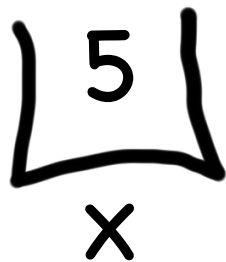
A hand-drawn diagram showing the variable `x` with a bracket above it containing the value 5.



A hand-drawn diagram showing the variable `y` with a bracket above it containing the value 5.

Tragedy Strikes

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



Noooooooooooooooooooo!

You see...

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

Since **y** is inside the if-block...

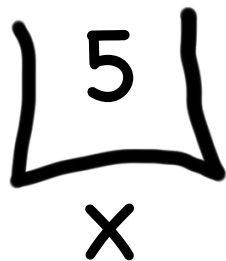
```
int x = 5;
```

```
if(lookingForLove()) {
```

```
    int y = 5;
```

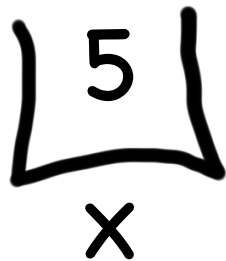
```
}
```

```
println(x + y);
```



...it goes away here...

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

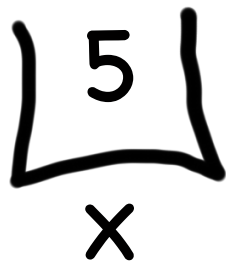


...and doesn't exist here.

```
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

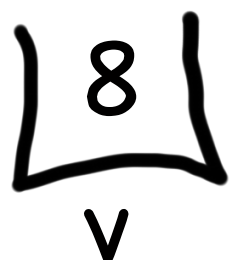
Variables have a lifetime (called scope):

```
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; ← Comes to life here  
    if (condition) {  
        v = 4;  
        ... some code  
    }  
    ... some other code  
}
```

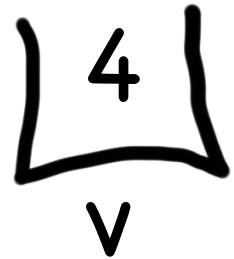


Variable Scope

Variables have a lifetime (called scope):

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

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

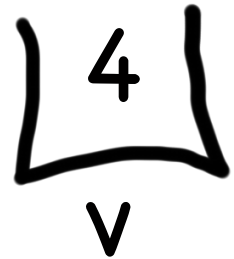


Variable Scope

Variables have a lifetime (called scope):

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

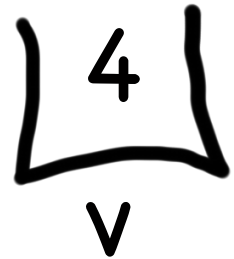
Still alive here...



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)

A Variable love story

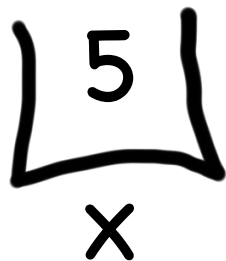
Chapter 2
By Chris

The programmer fixed their bug

...x was looking for love!

```
int x = 5;
```

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



A hand-drawn diagram consisting of a large, hand-drawn curly brace. Inside the top part of the brace is the number '5'. Below the bottom of the brace is the variable name 'x'. This diagram visually represents the state where the variable 'x' holds the value '5'.

...x was looking for love!

```
int x = 5;
```

```
if(lookingForLove()) {
```

```
    int y = 5;
```

```
    println(x + y);
```

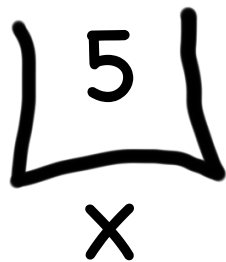
```
}
```

x was definitely
looking for love

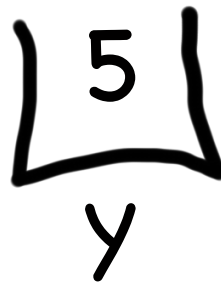
A hand-drawn diagram consisting of a large, hand-drawn curly brace. Inside the top part of the brace is the number '5'. Below the bottom of the brace is the variable name 'x'. This diagram visually represents the state of the variable x, which holds the value 5.

And met y.

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



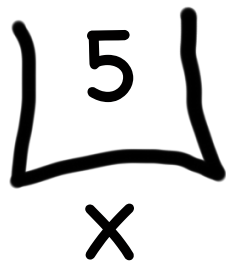
A hand-drawn diagram showing the variable `x` with a bracket above it containing the value 5.



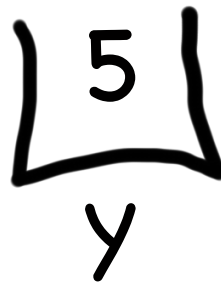
A hand-drawn diagram showing the variable `y` with a bracket above it containing the value 5.

Since they were both “in scope”...

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



A hand-drawn diagram showing the variable `x` with a bracket above it containing the value 5.



A hand-drawn diagram showing the variable `y` with a bracket above it containing the value 5.

...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 code block in which the variable was declared.
- A **code block** is a chunk of code between { } brackets

Recap

- Announcements
- Recap: Variables and Expressions
- Aside: Shorthand Operators + Constants
- Revisiting Control Flow
 - If and While
 - For
- Scope

Next time: Methods in Java