CS 106A, Lecture 5
Booleans and Control Flow

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
Plan For Today

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

• Everything canceled on Wednesday (7/4)
  – Lecture and LaIR are just not happening
  – Wednesday sections have been rescheduled
    • Go to your section leader’s if you can; otherwise, go to a different rescheduled section
• Assignment 1 due *Thursday at 11AM*
• Debugger tutorial up on website
Plan For Today

• Announcements

• Recap: Java, Variables and Expressions

• Aside: Shorthand Operators + Constants

• Revisiting Control Flow
  – If and While
  – For
Java

- Program
  - Karel Program
  - Console Program
    - SuperKarel Program
  - Graphics Program
• `println` allows output text to the user via the console
  – Output is the “O” in “I/O”

• We can also get input from the user via the console!
  – Use variables to store data collected via `readInt`, `readDouble`, etc.
• You can combine literals or variables together into *expressions* using binary operators:

+ Addition  
– Subtraction  
* Multiplication  
/ Division  
% Remainder
• When we divide integers, the quotient is also an integer.

  \[ 14 \div 4 = 3, \text{ not } 3.5. \]  \[(\text{Java ALWAYS rounds down.})\]

\[
\begin{array}{c}
4 \quad 3 \\
\underline{12} \\
\end{array}
\quad / 
\quad \begin{array}{c}
4 \quad 10 \\
\underline{40} \\
\end{array}
\quad 
\begin{array}{c}
52 \\
\underline{54} \\
\end{array}
\quad 27 \quad 1425
\]

\[
\begin{array}{c}
27 \quad 135 \\
\underline{135} \\
\end{array}
\quad 75
\]

\[
\begin{array}{c}
27 \quad 54 \\
\underline{54} \\
\end{array}
\quad 21
\]

• More examples:

  – \[ 32 \div 5 \quad \text{is } 6 \]
  \[ 32 \mod 5 \quad \text{is } 2 \]

  – \[ 84 \div 10 \quad \text{is } 8 \]
  \[ 84 \mod 10 \quad \text{is } 4 \]

  – \[ 156 \div 100 \quad \text{is } 1 \]
  \[ 156 \mod 100 \quad \text{is } 56 \]

– Dividing by 0 using / or % causes an error when your program runs.
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

e tc.

* The general rule is: operations always return the most expressive type
• **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 $11$
Practice

• \(1 / 2\) 0
• \(1.0 / 2\) 0.5
• \(1 + 2 / 3\) 1
• "abc" + (4 + 2) "abc6"
• "abc" + 4 + 2 "abc42"
Variable Types

**int** – an integer number

**double** – a decimal number

**char** – a single character

**boolean** – true or false
Making a new Variable

```cpp
int myVariable;
```
myVariable = 2;
• A variable can be declared_INITIALIZED in one statement.
  – This is probably the most commonly used declaration syntax.

• Syntax:

\[
\text{type name} = \text{expression};
\]

double tempF = 98.6;

int x = (12 / 2) + 3;
Plan For Today

• Announcements
• Recap: Variables and Expressions
• Aside: Shorthand Operators + Constants
• Revisiting Control Flow
  – If and While
  – For
# Shorthand Operators

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable += value;</td>
<td>variable = variable + value;</td>
</tr>
<tr>
<td>variable -= value;</td>
<td>variable = variable - value;</td>
</tr>
<tr>
<td>variable *= value;</td>
<td>variable = variable * value;</td>
</tr>
<tr>
<td>variable /= value;</td>
<td>variable = variable / value;</td>
</tr>
<tr>
<td>variable %= value;</td>
<td>variable = variable % value;</td>
</tr>
<tr>
<td>variable++;</td>
<td>variable = variable + 1;</td>
</tr>
<tr>
<td>variable--;</td>
<td>variable = variable - 1;</td>
</tr>
</tbody>
</table>

```plaintext
x -= 3;  // x = x - 3;
number *= 2;  // number = number * 2;
x++;  // x = x + 1;
```
• **constant**: A variable that cannot be changed after it is initialized. Declared at the top of your class, *outside of the run() method* but inside public class `Name`. 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:

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

– name is usually in ALL_UPPER_CASE

– Examples:

```java
private static final int DAYS_IN_WEEK = 7;
private static final double INTEREST_RATE = 3.5;
```
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);
    }
}
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
If/Else in Karel

```plaintext
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;
    statement;
    ...
}
```

Repeats the statements in the body until `condition` is no longer true. Each time, Karel executes all statements, and then checks the condition.
while (frontIsClear()) {
  body
}

if (beepersPresent()) {
  body
}
The condition should be a “boolean” which is either `true` or `false`
Booleans

1 < 2
Booleans

1 < 2

true
# Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equals</td>
<td><code>1 + 1 == 2</code></td>
<td>true</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>does not equal</td>
<td><code>3.2 != 2.5</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>10 &lt; 5</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>10 &gt; 5</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
<td><code>126 &lt;= 100</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
<td><code>5.0 &gt;= 5.0</code></td>
<td>true</td>
</tr>
</tbody>
</table>

* All have equal precedence
# Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equals</td>
<td><code>1 + 1 == 2</code></td>
<td>true</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>does not equal</td>
<td><code>3.2 != 2.5</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>10 &lt; 5</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>10 &gt; 5</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
<td><code>126 &lt;= 100</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
<td><code>5.0 &gt;= 5.0</code></td>
<td>true</td>
</tr>
</tbody>
</table>

* All have equal precedence
Relational Operators

```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.");
}
```
• **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

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

private static final int SENTINEL = -1;
(outside of run())

int sum = 0;
int num = readInt("Enter a number: ");
while (num != SENTINEL) {
    sum += num;
    num = readInt("Enter a number: ");
}
println("Sum is " + sum);
// Solution #2: "break" out of the loop
// ONLY appropriate to use in fencepost cases

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

Colin prefers this solution, but the debate of how to solve the “loop-and-a-half” problem has been raging for >50 years!
### Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>!(2 == 3)</td>
<td>true</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>(2 == 3) &amp;&amp; (-1 &lt; 5)</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
</tbody>
</table>

In order of precedence:

- `!` (not)
- `&&` (and)
- `||` (or)

They cannot be "chained" 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
// Store expressions that evaluate to true/false

boolean x = 1 < 2;    // true
boolean y = 5.0 == 4.0;  // false
// 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;
// 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?", "y", "n");
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.

![Image of program output](image-url)
Summary: Conditions

while (condition) {
    body
}

if (condition) {
    body
}

The condition should be a boolean which is either true or false
Plan For Today

• Announcements
• Recap: Variables and Expressions
• Aside: Shorthand Operators + Constants
• Revisiting Control Flow
  – If and While
  – For
for (int i = 0; i < max; i++) {
    statement;
    statement;
    ...
}

Repeats the statements in the body \textit{max} times.
For Loops in Java

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

- **This code is run once, just before the for loop starts.**
- **This code is run each time the code gets to the end of the ‘body’.**
- **Repeats the loop if this condition passes.**
For Loops in Java

```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!");
}
For Loops in Java

```java
for (int i = 0; i < 3; i++) {
    println("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!
```
For Loops in Java

for (int i = 0; i < 3; i++) {
    println("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!");
}
```

Output:
```
I love CS 106A!
```
For Loops in Java

```
for (int i = 0; i < 3; i++) {
    println("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!
For Loops in Java

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

Output:
```
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!
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!
i 3

```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!
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!
// prints the first 100 even numbers
for (int i = 0; i < 100; i++) {
    println(i * 2);
}
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);
// Launch countdown
for (int i = 10; i >= 1; i--) {
    println(i);
}
println("Blast off!");

Output:

10
9
8
...
1
Blast off!
Recap

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

Next time: More control flow, methods in Java
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.
```java
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");
}
```