Lecture 3
Decisions, Decisions

(C) Princeton University
The *For Loop*

Recall the *while* loop from FahrCels3:

```java
fahr=lower;
while(fahr<=upper) {
    .
    .
    .
    fahr=fahr+step;
}
```

This pattern is very common. In such cases, it is better to use a *for* loop

*General pattern:*

```java
for (i=begin;i<=end;i=i+incr){}
```

There are *three* parts inside the parens:

```java
for (fahr=lower;fahr<=upper;fahr=fahr+step) {
    .
    an initialization statement: i = begin
}
```

- a logical condition: \( i < \text{end} \)
- an index update expression: \( i = i+\text{incr} \)

Note: there are only *two* semicolons.
More on For

Recall the basic pattern:

```c
for (i=begin;i<end;i=i+incr) {
   ...
}
```

Variable `i` is called the *looping index*. The looping index should always be an integer.

The symbol `++` is called the *increment operator*. It means add 1 to the variable to which it is attached. Right here. Right now.

This loop is by far the most common loop construct.
### Computing Square Roots

#### The algorithm:

<table>
<thead>
<tr>
<th>Given: a number ( x )</th>
<th>Call that number ( s ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guess a value for ( \sqrt{x} ), say ( x/2 )</td>
<td>If ( r &gt; \sqrt{x} ), then ( s &lt; \sqrt{x} ), and vice versa.</td>
</tr>
<tr>
<td>Call it ( r ).</td>
<td>Okay, average the two and call that ( r : r \leftarrow \frac{r + s}{2} )</td>
</tr>
<tr>
<td>Divide it into ( x ).</td>
<td>Repeat these steps until ( r^2 \approx x )</td>
</tr>
</tbody>
</table>

#### Using a **do-while** loop

```
r = x/2;
do {
s = x/r;
r = (r+s)/2;
err = Math.abs(x-r*r);
} while (err > EPSILON);
```

In Java:

```
r = x/2;
err = Math.abs(x - r*r);
while (err > EPSILON) {
s = x/r;
r = (r+s)/2;
err = Math.abs(x - r*r);
}
```

In this case, a **do-while** loop is better!
import java.text.DecimalFormat;
public class Sqrt
{
    static final double EPSILON = 1.0e-10;
    public static void main(String[] args)
    {
        int i;
        double x,r,s,err;
        DecimalFormat xfmt = new DecimalFormat("000.#");
        DecimalFormat xfmt = new DecimalFormat("000.#####");
        for (i=1; i<20; i++) {
            x = i;
            r = x/2;
            do {
                s = x/r;
                r = (r+s)/2;
                err = Math.abs(x - r*r);
            } while (err > EPSILON);
            System.out.print(xfmt.format(x) + "	" + xfmt.format(r));
        }
    }
}
Here’s the Output

<table>
<thead>
<tr>
<th>1.0</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>1.41421</td>
</tr>
<tr>
<td>3.0</td>
<td>1.73205</td>
</tr>
<tr>
<td>4.0</td>
<td>2.0</td>
</tr>
<tr>
<td>5.0</td>
<td>2.23607</td>
</tr>
<tr>
<td>6.0</td>
<td>2.44949</td>
</tr>
<tr>
<td>7.0</td>
<td>2.64575</td>
</tr>
<tr>
<td>8.0</td>
<td>2.82843</td>
</tr>
<tr>
<td>9.0</td>
<td>3.0</td>
</tr>
<tr>
<td>10.0</td>
<td>3.16228</td>
</tr>
<tr>
<td>11.0</td>
<td>3.31662</td>
</tr>
<tr>
<td>12.0</td>
<td>3.4641</td>
</tr>
<tr>
<td>13.0</td>
<td>3.60555</td>
</tr>
<tr>
<td>14.0</td>
<td>3.74116</td>
</tr>
<tr>
<td>15.0</td>
<td>3.87298</td>
</tr>
<tr>
<td>16.0</td>
<td>4.0</td>
</tr>
<tr>
<td>17.0</td>
<td>4.12311</td>
</tr>
<tr>
<td>18.0</td>
<td>4.24264</td>
</tr>
<tr>
<td>19.0</td>
<td>4.3589</td>
</tr>
</tbody>
</table>

"A wonderful square root. Let’s hope it can be used for the good of mankind."