While Loops

The while loop. A common repetition structure.
- Check a boolean expression.
- Execute a sequence of statements.
- Repeat.

```
while (boolean expression) {
    statement 1;
    statement 2;
}
```

Ex. Print first $n$ powers of 2.
- Increment $i$ from 1 to $n$.
- Double $v$ each time.

```
int i = 0;
int v = 1;
while (i <= n) {
    System.out.println(v);
    i = i + 1;
    v = 2 * v;
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>v</th>
<th>i &lt;= n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>true</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>true</td>
</tr>
</tbody>
</table>

$n = 6$
**Powers of Two**

```java
class PowersOfTwo {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int i = 0;
        int v = 1;
        while (i <= 6) {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```

```java
class Sqrt {
    public static void main(String[] args) {
        double EPS = 1E-15;
        double t = Double.parseDouble(args[0]);
        double c = 6;
        while (Math.abs(t - c / t) > t * EPS) {
            t = (c / t + t) / 2.0;
        }
        System.out.println(t);
    }
}
```

**While Loops: Square Root**

**Q.** How might we implement `Math.sqrt()`?

**A.** To compute the square root of `c`:

- Initialize `t0 = c`.
- **Repeat until** `|t_i - t_{i-1}| < error tolerance` up to desired precision:
  - set `t_{i+1}` to be the average of `t_i` and `c / t_i`.

**Newton-Raphson Method**

**Square root method explained.**

- **Goal:** find root of function `f(x)`.
- **Start with estimate** `t0`.
  - **Draw line tangent to curve at x = t0.**
  - **Set** `t_{i+1}` to be x-coordinate where line hits x-axis.
  - **Repeat until desired precision.**

**Computation of the square root of 2:**

\[
\begin{align*}
t_0 &= 2.0 \\
t_1 &= \frac{1}{2}(t_0 + \frac{2}{t_0}) = 1.5 \\
t_2 &= \frac{1}{2}(t_1 + \frac{2}{t_1}) = 1.4166666666666665 \\
t_3 &= \frac{1}{2}(t_2 + \frac{2}{t_2}) = 1.414215682745097 \\
t_4 &= \frac{1}{2}(t_3 + \frac{2}{t_3}) = 1.414213562374899 \\
t_5 &= \frac{1}{2}(t_4 + \frac{2}{t_4}) = 1.414213562373095
\end{align*}
\]

15 decimal digits of accuracy in 5 iterations
**The For Loop**

**For Loops: Subdivisions of a Ruler**

*Create subdivision of a ruler.*
- Initialize `ruler` to empty string.
- For each value `i` from 1 to `N`:
  - sandwich two copies of `ruler` on either side of `i`.

```java
int N = 3;
String ruler = " ";
for (int i = 1; i <= N; i++) {
    ruler = ruler + i + ruler;
}
System.out.println(ruler);
```

<table>
<thead>
<tr>
<th>i</th>
<th>ruler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot; 1 &quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot; 1 2 1 &quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot; 1 2 1 3 1 2 1 &quot;</td>
</tr>
</tbody>
</table>

**Observation.** Loops can produce a huge amount of output!

---

**The for loop.** Another common repetition structure.
- Execute initialization statement.
- Check boolean expression.
- Execute sequence of statements.
- Execute increment statement.
- Repeat.

```java
for (init; boolean expression; increment) {
    statement 1;
    statement 2;
}
```

**For Loops: Subdivisions of a Ruler**

```java
% java Ruler 1
1
% java Ruler 2
1 2 1
% java Ruler 3
1 2 1 3 1 2 1
% java Ruler 4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
% java Ruler 5
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
% java Ruler 100
Exception in thread "main"
java.lang.OutOfMemoryError
```
Nesting Conditionals and Loops

**Conditionals** enable you to do one of $2^n$ sequences of operations with $n$ lines.

```java
if (a0 > 0) System.out.println(0);
if (a1 > 0) System.out.println(1);
if (a2 > 0) System.out.println(2);
if (a3 > 0) System.out.println(3);
if (a4 > 0) System.out.println(4);
if (a5 > 0) System.out.println(5);
if (a6 > 0) System.out.println(6);
if (a7 > 0) System.out.println(7);
if (a8 > 0) System.out.println(8);
if (a9 > 0) System.out.println(9);
```

Loops enable you to do an operation $n$ times using only 2 lines of code.

```java
double sum = 0.0;
for (int i = 1; i <= 1024; i++)
    sum = sum + 1.0 / i;
```

$2^n = 1024$ possible results, depending on input

More sophisticated programs.
- Nest conditionals within conditionals.
- Nest loops within loops.
- Nest conditionals within loops within loops.

Gambler’s Ruin

**Gambler’s ruin.** Gambler starts with $\text{stake}$ and places $\$1$ fair bets until going broke or reaching $\text{goal}$.
- What are the chances of winning?
- How many bets will it take?

**One approach.** Monte Carlo simulation.
- Flip digital coins and see what happens.
- Repeat and compute statistics.

---

**Nested If-Else**

**Ex.** Pay a certain tax rate depending on income level.

<table>
<thead>
<tr>
<th>Income</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47,450</td>
<td>22%</td>
</tr>
<tr>
<td>47,450 - 114,650</td>
<td>25%</td>
</tr>
<tr>
<td>114,650 - 174,700</td>
<td>28%</td>
</tr>
<tr>
<td>174,700 - 311,950</td>
<td>33%</td>
</tr>
<tr>
<td>311,950 -</td>
<td>35%</td>
</tr>
</tbody>
</table>

```java
double rate = 0.35;
if (income < 47450) rate = 0.22;
if (income < 114650) rate = 0.25;
if (income < 174700) rate = 0.28;
if (income < 311950) rate = 0.33;
```

Wrong graduated income tax calculation
Debugging

```java
public class Gambler {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);
        int wins = 0;
        // repeat experiment N times
        for (int i = 0; i < trials; i++) {
            int t = stake;
            while (t > 0 && t < goal) {
                // flip coin and update
                if (Math.random() < 0.5) t++;
                else t--;
                if (t == goal) wins++;
            }
        }
        System.out.println(wins + " wins of " + trials);
    }
}
```

Simulation and Analysis

Fact. Probability of winning = stake ÷ goal.
Fact. Expected number of bets = stake × goal.
Ex. 20% chance of turning $500 into $2500, but expect to make one million $1 bets.

Remark. Both facts can be proved mathematically; for more complex scenarios, computer simulation is often the best plan of attack.

Debugging a Program

Factor. Given an integer N, compute its prime factorization.

```
3,757,208 = 2³ × 7 × 13² × 397
```

Application. Break RSA cryptosystem.
Debugging a Program: Syntax Errors

Syntax error. Illegal Java program.

- Compiler error messages help locate problem.
- Eventually, a file named Factors.class.

```
public class Factors1 {   public static void main(String[] args) {      long N = Long.parseLong(args[0]);      for (i = 0; i < N; i++) {         System.out.print(i + " ");      }   }
```

- Check if i is a factor.
- As long as i is a factor, divide it out.
- Corner case: biggest factor occurs once.

```
public class Factors2 {   public static void main(String[] args) {      long N = Long.parseLong(args[0]);      for (long i = 2; i <= N; i++) {         System.out.print(i + " ");         N = N / i;      }   }
```

- Check if i is a factor.
- As long as i is a factor, divide it out.
- No output (17) or infinite loop (49)

```
public class Factors3 {   public static void main(String[] args) {      long N = Long.parseLong(args[0]);      for (long i = 2; i <= N; i++) {         System.out.print(i + " ");         N = N / i;      }   }
```

- Check if i is a factor.
- As long as i is a factor, divide it out.
- Corner case: biggest factor occurs once.

Debugging a Program: Performance Errors

Performance error. Correct program but too slow.

- Use profiling to discover bottleneck.
- Devise better algorithm.

```
public class Factors {   public static void main(String[] args) {      long N = Long.parseLong(args[0]);      for (long i = 2; i <= N; i++) {         System.out.print(i + " ");         N = N / i;      }   }
```

- As long as i is a factor, divide it out.
- Check if i is a factor.
- Too slow for large N (999,999,937)

```
if (N > 1) System.out.println(N); else System.out.println();
```

- Check if i is a factor.
- As long as i is a factor, divide it out.
- Corner case: biggest factor occurs once.

Fact. If N has a factor, it has one less than or equal to its square root.

Impact. Many fewer iterations of for loop.

Debugging a Program: Semantic Errors

Semantic error. Legal but wrong Java program.

- Use "System.out.println" method to identify problem.
Debugging a Program: Analysis

**Q. How large an integer can I factor?**

<table>
<thead>
<tr>
<th>Digits</th>
<th>$\leq N$</th>
<th>$\leq 1 \leq N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>6</td>
<td>0.15 seconds</td>
<td>instant</td>
</tr>
<tr>
<td>9</td>
<td>77 seconds</td>
<td>instant</td>
</tr>
<tr>
<td>12</td>
<td>21 hours†</td>
<td>0.16 seconds</td>
</tr>
<tr>
<td>15</td>
<td>2.4 years†</td>
<td>2.7 seconds</td>
</tr>
<tr>
<td>18</td>
<td>2.4 millennia†</td>
<td>92 seconds</td>
</tr>
</tbody>
</table>

† estimated

Debugging. Cyclic process of editing, compiling, and fixing errors.
- Always a logical explanation.
- What would the machine do?
- Explain it to the teddy bear.

You will make many mistakes as you write programs. It’s normal.

As soon as we started programming, we found out to our surprise that it wasn’t as easy to get programs right as we had thought. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs. - Maurice Wilkes

If I had 8 hours to chop down a tree, I would spend 6 hours sharpening an axe. - Anonymous

Programming in Java

Programming in Java. [a slightly more realistic view]

1. Create the program.

2. Compile it.
   - Compiler says: That’s not a legal program.
   - Back to step 1 to fix your errors of syntax.

3. Execute it.
   - Result is bizarrely (or subtly) wrong.
   - Back to step 1 to fix your errors of semantics.

4. Enjoy the satisfaction of a working program!

Control Flow Summary

Control flow.
- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enables us to choreograph the control flow.

<table>
<thead>
<tr>
<th>Control Flow</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight-line programs</td>
<td>All statements are executed in the order given.</td>
<td></td>
</tr>
<tr>
<td>Conditionals</td>
<td>Certain statements are executed depending on the values of certain variables.</td>
<td>if if-else</td>
</tr>
<tr>
<td>Loops</td>
<td>Certain statements are executed repeatedly until certain conditions are met.</td>
<td>while for do-while</td>
</tr>
</tbody>
</table>