3. Conditionals and loops
3. Conditionals & Loops

- **Conditionals**: the *if* statement
- **Loops**: the *while* statement
- An alternative: the *for* loop
- **Nesting**
- **Debugging**
Context: basic building blocks for programming

Any program you might want to write can be built from the following basic components:

- **Objects**
- **Functions and modules**
- **Graphics, sound, and image I/O**
- **Arrays**
- **Conditionals and loops**
- **Math**
- **Text I/O**
- **Primitive data types**
- **Assignment statements**

Previous lecture: equivalent to a calculator

This lecture: to infinity and beyond!
Conditionals and Loops

Control flow
- The sequence of statements that are actually executed in a program.
- **Conditionals and loops** enable us to choreograph control flow.

straight-line control flow  [ previous lecture ]

control flow with conditionals and a loop  [this lecture]
The if statement

Execute certain statements depending on the values of certain variables.
- Evaluate a boolean expression.
- If true, execute a statement.
- The else option: If false, execute a different statement.

Example: if (x < 0) x = -x;
Computes the absolute value of x

Example: if (x > y) max = x; else max = y;
Computes the maximum of x and y
Example of if statement use: simulate a coin flip

```java
public class Flip
{
    public static void main(String[] args)
    {
        if (Math.random() < 0.5)
            System.out.println("Heads");
        else System.out.println("Tails");
    }
}
```

% java Flip
Heads
% java Flip
Heads
% java Flip
Tails
% java Flip
Heads
Example of if statement use: 2-sort

**Q.** What does this program do?

```java
public class TwoSort
{
    public static void main(String[] args)
    {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        if (b < a)
        {
            int t = a;  
            a = b;
            b = t;
        }
        StdOut.println(a);
        StdOut.println(b);
    }
}
```

% java TwoSort 1234 99
99
1234

% java TwoSort 99 1234
99
1234

**A.** Reads two integers from the command line, then prints them out in numerical order.
Pop quiz on if statements

Q. Add code to this program that puts a, b, and c in numerical order.

public class ThreeSort
{
    public static void main(String[] args)
    {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int c = Integer.parseInt(args[2]);

        StdOut.println(a);
        StdOut.println(b);
        StdOut.println(c);
    }
}

% java ThreeSort 1234 99 1
1
99
1234

% java ThreeSort 99 1 1234
1
99
1234
Example of if statement use: error checks

```java
public class IntOps {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int sum = a + b;
        int prod = a * b;
        System.out.println(a + " + " + b + " = " + sum);
        System.out.println(a + " * " + b + " = " + prod);
        if (b == 0) System.out.println("Division by zero");
        else System.out.println(a + " / " + b + " = " + a / b);
        if (b == 0) System.out.println("Division by zero");
        else System.out.println(a + " % " + b + " = " + a % b);
    }
}
```

Good programming practice. Use conditionals to check for and avoid runtime errors.
3. Conditionals & Loops

- Conditionals: the `if` statement
- **Loops**: the `while` statement
- An alternative: the `for` loop
- Nesting
- Debugging
The **while loop**

Execute certain statements repeatedly until certain conditions are met.

- Evaluate a boolean expression.
- If true, execute a sequence of statements.
- Repeat.

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

Prints the powers of two from $2^0$ to $2^n$.

[stay tuned for a trace]
Example of while loop use: print powers of two

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

Prints the powers of two from $2^0$ to $2^n$.

<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>false</td>
</tr>
</tbody>
</table>

```
% java PowersOfTwo 6
1
2
4
8
16
32
64
```
Pop quiz on while loops

Q. Anything wrong with the following code?

```java
public class PQwhile {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int i = 0;
        int v = 1;
        while (i <= n) {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```
Example of while loop use: implement Math.sqrt()

Goal. Implement square root function.

Newton-Raphson method to compute $\sqrt{c}$

- Initialize $t_0 = c$.
- Repeat until $t_i = c/t_i$ (up to desired precision):
  - Set $t_{i+1}$ to be the average of $t_i$ and $c/t_i$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$t_i$</th>
<th>$2/t_i$</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.0</td>
<td>1.0</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>1.5</td>
<td>1.3333333</td>
<td>1.4166667</td>
</tr>
<tr>
<td>2</td>
<td>1.4166667</td>
<td>1.4117647</td>
<td>1.4142157</td>
</tr>
<tr>
<td>3</td>
<td>1.4142157</td>
<td>1.4142114</td>
<td>1.4142136</td>
</tr>
<tr>
<td>4</td>
<td>1.4142136</td>
<td>1.4142136</td>
<td></td>
</tr>
</tbody>
</table>

computing the square root of 2 to seven places

% java Sqrt 60481729
7777.0
% java Sqrt 2
1.4142136

The square root of 9 is 3.
A) True.
B) False.
C) Who cares?

Many students actually look forward to Mr. Atwadder’s math tests.
Example of while loop use: implement `Math.sqrt()`

Newton-Raphson method to compute $\sqrt{c}$

- Initialize $t_0 = c$.
- Repeat until $t_i = c/t_i$ (up to desired precision):
  - Set $t_{i+1}$ to be the average of $t_i$ and $c / t_i$.

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

Scientists studied computation well before the onset of the computer.

Isaac Newton
1642-1727

% java Sqrt 60481729
7777.0

% java Sqrt 2.0
1.414213562373095
Newton-Raphson method

**Explanation** (some math omitted)

- **Goal**: find root of function $f(x)$. Use $f(x) = x^2 - c$ for $\sqrt{c}$
- **Start with estimate** $t_0$.
- **Draw line tangent** to curve at $x = t_i$.
- **Set** $t_{i+1}$ **to be** $x$-coordinate **where line hits** $x$-**axis**.
- **Repeat** until desired precision.
3. Conditionals & Loops

• Conditionals: the if statement
• Loops: the while statement
• An alternative: the for loop
• Nesting
• Debugging
The for loop

An alternative repetition structure.
- Evaluate an *initialization statement*.
- Evaluate a boolean expression.
- If true, execute a sequence of statements, then execute an *increment statement*.
- Repeat.

Example:
```java
int v = 1;
for (int i = 0; i <= n; i++)
{
    System.out.println ( i + " " + v );
    v = 2*v;
}
```

Prints the powers of two from $2^0$ to $2^n$

Why? Can provide code that is more compact and understandable.

Every for loop has an equivalent while loop:
```java
int v = 1;
int i = 0;
while (i <= n)
{
    System.out.println ( i + " " + v );
    v = 2*v;
    i++;
}
```
Examples of for loop use

```java
int sum = 0;
for (int i = 1; i <= N; i++)
    sum += i;
System.out.println(sum);
```

Compute sum \(1 + 2 + 3 + \ldots + N\)

<table>
<thead>
<tr>
<th>i</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

```java
long product = 1;
for (int i = 1; i <= N; i++)
    product *= i;
System.out.println(product);
```

Compute \(N! (1 * 2 * 3 * \ldots * N)\)

<table>
<thead>
<tr>
<th>i</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
</tbody>
</table>

```java
for (int k = 0; k <= N; k++)
    System.out.println(k + " " + 2*Math.PI*k/N);
```

Print a table of function values

<table>
<thead>
<tr>
<th>k</th>
<th>(\frac{2\pi k}{N})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>1.57079632...</td>
</tr>
<tr>
<td>2</td>
<td>3.14159265...</td>
</tr>
<tr>
<td>3</td>
<td>4.71238898...</td>
</tr>
<tr>
<td>4</td>
<td>6.28318530...</td>
</tr>
</tbody>
</table>

```java
int v = 1;
while (v <= N/2)
    v = 2*v;
System.out.println(v);
```

Print largest power of 2 less than or equal to \(N\)

<table>
<thead>
<tr>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>16</td>
</tr>
</tbody>
</table>
Example of for loop use: subdivisions of a ruler

Create subdivisions of a ruler to $1/N$ inches.

- Initialize \texttt{ruler} to one space.
- For each value $i$ from 1 to $N$:
  sandwich $i$ between two copies of \texttt{ruler}.

```
public class Ruler {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        String ruler = " ";
        for (int i = 1; i <= N; i++)
            ruler = ruler + i + ruler;
        System.out.println(ruler);
    }
}
```

Note: Small program can produce huge amount of output.

End-of-loop trace

\begin{tabular}{|c|}
\hline
i & \textbf{ruler} \\
\hline
1 & " 1 " \\
2 & " 1 2 1 " \\
3 & " 1 2 1 3 1 2 1 " \\
4 & " 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 " \\
\hline
\end{tabular}

java Ruler 4
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

$2^{100} - 1$ integers in output (!)
```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```
Pop quiz on for loops (easy if you read exercise 1.3.13)

Q. What does the following program print?

```java
public class PQfor {
    public static void main(String[] args) {
        int f = 0, g = 1;
        for (int i = 0; i <= 10; i++) {
            System.out.println(f);
            f = f + g;
            g = f - g;
        }
    }
}
```
3. Conditionals & Loops

- Conditionals: the if statement
- Loops: the while statement
- An alternative: the for loop
- Nesting
- Debugging
Nesting conditionals and loops

Nesting

• Any “statement” within a conditional or loop may itself be a conditional or a loop statement.
• Enables complex control flows.
• Adds to challenge of debugging.

Example:

```java
for (int i = 0; i < trials; i++)
{
    int t = stake;
    while (t > 0 && t < goal)
    {
        if (Math.random() < 0.5) t++;
        else t--;
        if (t == goal) wins++;
    }
```

[Stay tuned for an explanation of this code.]
Example of nesting conditionals: Tax rate calculation

**Goal.** Given income, calculate proper tax rate.

```
if (income < 47450) rate = 0.22;
else {
    if (income < 114650) rate = 0.25;
    else {
        if (income < 174700) rate = 0.28;
        else {
            if (income < 311950) rate = 0.33;
            else rate = 0.35;
        }
    }
}
```

<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 – $114,649</td>
<td>25%</td>
</tr>
<tr>
<td>$114,650 – $174,699</td>
<td>28%</td>
</tr>
<tr>
<td>$174,700 – $311,949</td>
<td>33%</td>
</tr>
<tr>
<td>$311,950 +</td>
<td>35%</td>
</tr>
</tbody>
</table>
Pop quiz on nested if statements

Q. Anything wrong with the following code?

```java
public class PQif
{
    public static void main(String[] args)
    {
        double income = Double.parseDouble(args[0]);
        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;
        System.out.println(rate);
    }
}
```
Gambler's ruin problem

A gambler starts with $stake and places $1 fair bets.
- Outcome 1 (loss): Gambler goes broke with $0.
- Outcome 2 (win): Gambler reaches $goal.

Q. What are the chances of winning?
Q. How many bets until win or loss?

One approach: Monte Carlo simulation.
- Use a simulated coin flip.
- Repeat and compute statistics.
Example of nesting conditionals and loops: Simulate gambler's ruin

Gambler's ruin simulation

- Get command-line parms.
- Run all the experiments.
- Run one experiment.
- Make one bet.
- If goal met, count the win.
- Print #wins and # trials.

```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;
        for (int i = 0; i < trials; i++) {
            int t = stake;
            while (t > 0 && t < goal) {
                if (Math.random() < 0.5) t++;
                else t--;
            }
            if (t == goal) wins++;
        }
        StdOut.println(wins + " wins of " + trials);
    }
}
```

% java Gambler 5 25 1000
203 wins of 1000
Digression: simulation and analysis

Facts (known via mathematical analysis for centuries)
- Probability of winning = stake ÷ goal.
- Expected number of bets = stake × desired gain.

Example
- 20% chance of turning $500 into $2500. 
  500/2500 = 20%
- Expect to make 1 million $1 bets.
  500*(2500 - 500) = 1,000,000

Remarks
- Computer simulation can help validate mathematical analysis.
- For this problem, mathematical analysis is simpler (if you know the math).
- For more complicated variants, computer simulation may be the best plan of attack.
3. Conditionals & Loops

- Conditionals: the if statement
- Loops: the while statement
- An alternative: the for loop
- Nesting
- Debugging
Debugging is 99% of program development in any programming language, even for experts.

**Bug:** A mistake in a program.

**Debugging:** The process of eliminating bugs.

"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

**Impossible ideal:** "Please compile, execute, and debug my program."

**Bottom line:** Programming is primarily a *process* of finding and fixing mistakes.
Debugging

is challenging because conditionals and loops *dramatically increase* the number of possible outcomes.

<table>
<thead>
<tr>
<th>program structure</th>
<th>no loops</th>
<th>$N$ conditionals</th>
<th>1 loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of possible execution sequences</td>
<td>1</td>
<td>$2^N$</td>
<td>no limit</td>
</tr>
</tbody>
</table>

Most programs contain *numerous* conditionals and loops, with nesting.

**Good news.** Conditionals and loops provide structure that helps us understand our programs.

Old and low-level languages have a *goto* statement that provides arbitrary structure. Eliminating *gos* was controversial until Edsger Dijkstra published the famous note "*Goto considered harmful*" in 1968.

“The quality of programmers is a decreasing function of the number of goto statements in the programs they produce.”

— Edsger Dijkstra
Debugging a program: a running example

**Problem:** Factor a large integer $N$.

**Application:** Cryptography.

**Suprising fact:** Security of internet commerce depends on difficulty of factoring large integers.

**Method**

- Consider each integer $i$ less than $N$
- While $i$ divides $N$ evenly
  - Print $i$ (it is a factor of $N$).
  - Replace $N$ with $N/i$.

**Rationale:**
1. Any factor of $N/i$ is a factor of $N$.
2. $i$ may be a factor of $N/i$.

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

This program has bugs!
Debugging a program: syntax errors

**Is your program a legal Java program?**

- Java compiler can help you find out.
- Use javac to find the first error.
- Repeat.
- Result: An executable Factors.class file

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

% javac Factors.java
Factors.java:5: ';' expected
    long N = Long.parseLong(args[0]);
          ^

% javac Factors.java
Factors.java:6: cannot find symbol
   symbol : variable i
   location: class Factors
     for ( int i = 0; i < N; i++)
           ^

% javac Factors.java
%
Debugging a program: runtime and semantic errors

Does your legal Java program do what you want it to do?
• You need to run it to find out.
• Use java runtime to find the first error.
• Fix and repeat.

% javac Factors.java
% java Factors  oops, need argument
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
  at Factors.main(Factors.java:5)

% java Factors 98
Exception in thread "main"
java.lang.ArithmeticException: / by zero
  at Factors.main(Factors.java:8)

% java Factors 98
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

% java Factors 98
  2 7 7%

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

This working program still has bugs!
Debugging a program: testing

Does your legal Java program *always* do what you want it to do?

• You need to test on many types of inputs it to find out.
• Add trace code to find the first error.
• Fix the error.
• Repeat.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++)
            if (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        System.out.println("TRACE " + i + " " + N);
    }
}
```
Debugging a program: testing

Does your legal Java program always do what you want it to do?
- You need to test on many types of inputs it to find out.
- Add trace code to find the first error.
- Fix the error.
- Repeat.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            while (N % i == 0)
                { System.out.print(i + " ");
                    N = N / i; }
        }
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

Note: This working program still has a bug (stay tuned).
Debugging a program: performance

Is your working Java program fast enough to solve your problem?
- You need to test it on increasing problem sizes to find out.
- May need to change the algorithm to fix it.
- Repeat.

Method
- Consider each integer \( i \leq N/i \)
- While \( i \) divides \( N \) evenly
  - print \( i \) (it is a factor of \( N \))
  - replace \( N \) with \( N/i \).

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N/i; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

might work, but way too slow
change the _algorithm_: no need to check when \( i \cdot i > N \) since all smaller factors already checked
implement the change

```
% java Factors
11 73 101 137
% java Factors
21649 513239
% java Factors
11239 4649 909091
% java Factors
2071723 5363222357
```
Debugging a program: performance analysis

Q. How large an integer can I factor?

public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i <= N/i; i++)
            {  
                while (N % i == 0)
                {  
                    System.out.print(i + " ");
                    N = N / i;
                }
        }
    if (N > 1) System.out.println(N);
    else System.out.println();
    }
}

<table>
<thead>
<tr>
<th>digits in largest factor</th>
<th>i &lt; N</th>
<th>i &lt;= N/i</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>6</td>
<td>instant</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>instant</td>
</tr>
<tr>
<td>15</td>
<td>2.4 years†</td>
<td>2.7 seconds</td>
</tr>
<tr>
<td>18</td>
<td>2.4 millenia†</td>
<td>92 seconds</td>
</tr>
</tbody>
</table>

† estimated, using analytic number theory

Lesson. Performance matters!

Note. Internet commerce is still secure: it depends on the difficulty of factoring 200-digit integers.
Debugging your program: summary

Program development is a *four*-step process, with feedback.

**EDIT** your program.

**COMPILE** your program to create an executable file.

**RUN** your program to test that it works as you imagined.

**TEST** your program on realistic and real input data.

**SUBMIT** your program for independent testing and approval.

- syntax error
- runtime error
- semantic error
- performance error

Telling a computer what to do when you know what you're doing
3. Conditionals & Loops