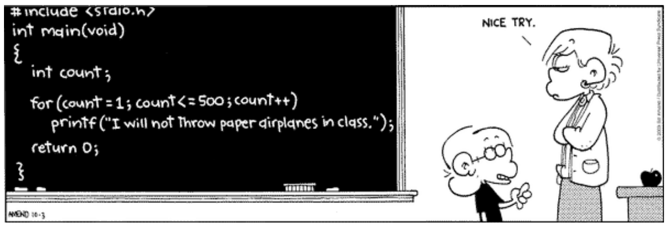
Lecture 3: Loops



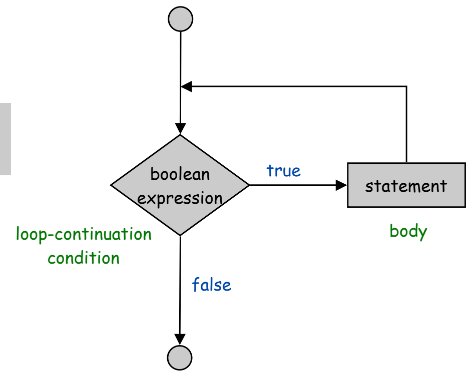
While Loops

The `while` loop is a common repetition structure.

- Check loop-continuation condition.
- Execute a sequence of statements.
- Repeat.

```
while (boolean expression)
   statement;
```

while loop syntax



while loop flow chart

While Loops: Powers of Two

While loop example: print powers of 2.

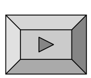
- Increment `i` from 1 to 6 by 1.
- Double `N` each time.

```
int i = 0;
int N = 1;
while (i <= 6) {
   System.out.println(N);
   i = i + 1;
   N = 2 * N;
}
```

a block statement

i	N	i <= 6
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	true

```
% java Powers
1
2
4
8
16
32
64
```

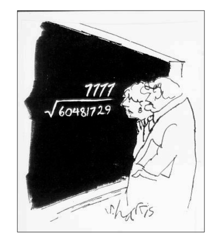


Click for demo

While Loops: Newton-Raphson Method

How might we implement `Math.sqrt` ?

- Goal: compute the square root of c.
- Initialize t = c.
- Replace t with the average of t and c / t.
- Repeat until t = c / t, up to desired precision.



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


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

error tolerance

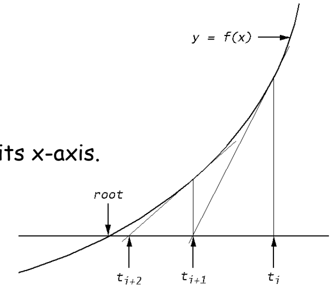
```
% java Sqrt 2.0
1.414213562373095
```

15 decimal digits of accuracy in 5 iterations

While Loops: Newton-Raphson Method

Newton-Raphson method explained.

- Goal: find root of function f(x).
  - Ex: $f(x) = x^2 - c$
- 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.



Applications and extensions.

- Find the roots of a differentiable function of one variable.
- Find the roots of a function of several variables.
- Optimize a twice differentiable function: find where derivative = 0.
- Optimize a function subject to constraints.

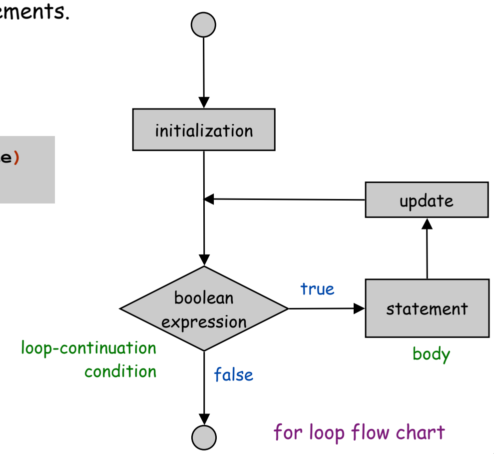
For Loops

The `for` loop is another common repetition structure.

- Initialize variable.
- Check loop-continuation condition.
- Execute sequence of statements.
- Increment variable.
- Repeat.

```
for (init; boolean; update)
   statement;
```

for loop syntax



for loop flow chart

For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

- Initialize ruler to the empty string.
- For each value i = 1 to N.
- Sandwich two copies of the ruler on either side of i.

```
String ruler = " ";

for (int i = 1; i <= N; i++) {
   ruler = ruler + i + ruler;
}

System.out.println(ruler);
```

Java code

i	*ruler*
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

For Loops: Subdivisions of a Ruler

Observation.

- Program produces $2^N - 1$ integers.
- Loops can produce a huge amount of output!

```
% 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 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 of code.

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

1024 possible results, depending on input

Loops enable you to do something N times using only 2 lines of code.

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

computes 1/1 + 1/2 + ... + 1/1024

More sophisticated programs.

- Nest conditionals within conditionals.
- Nest loops within loops.
- Nest conditionals within loops within loops.

Nested If-Else

Nesting conditionals within conditionals.

- Ex: Pay a certain tax rate depending on income level.

```
double 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;
```

graduated income tax calculation

Income	*Rate*
0 - 47,450	22%
47,450 - 114,650	25%
114,650 - 174,700	28%
174,700 - 311,950	33%
311,950 -	35%

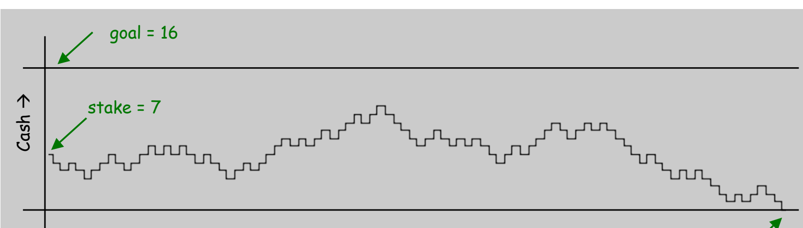
Gambler's Ruin

Gambler starts with $stake and places $1 even bets until going broke or reaching $goal.

- What are the chances of winning?
- How many bets will it take?

One approach: numerical simulation.

- Flip digital coins and see what happens.
- Repeat and compute statistics.



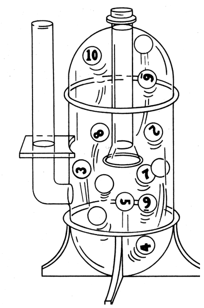
One simulation of the gambler's ruin problem.

Library Functions: Math.random

`Math.random` generates number between 0 and 1.

How is `Math.random` implemented?

- Linear feedback shift register? Cosmic rays?
- User doesn't need to know details.
- User doesn't want to know details.



Caveats.

- "Random" numbers are not really random.
- Don't use for crypto or Internet gambling!
- Check assumptions about library function before using.

Gambler's Ruin

```
public class Gambler {
   public static void main(String[] args) {
      int stake = Integer.parseInt(args[0]);
      int goal  = Integer.parseInt(args[1]);
      int N     = Integer.parseInt(args[2]);
      int wins  = 0;

      // repeat simulation N times
      for (int i = 0; i < N; i++) {

         // do gambler's ruin simulation
         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 " + N);
   }
}
```


Simulation and Analysis

stake goal N

```
% java Gambler 10 20 1000
513 wins of 1000

% java Gambler 10 20 1000
492 wins of 1000

% java Gambler 500 2500 100
24 wins of 100
```

takes a few minutes

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

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

Debugging a Program: Syntax Errors

Given an integer N, compute its prime factorization. $168 = 2^3 \times 3 \times 7$

- Application: break RSA cryptosystem.

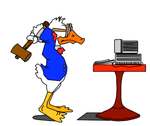
Syntax error: illegal Java program.

- 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++) {
         while (N % i == 0)
            System.out.print(i + " ")
            N = N / i
      }
   }
}
```

Check if i is a factor.

As long as i is a factor, divide it out.



Does not compile

Debugging a Program: Semantic Errors

Semantic error: legal but wrong Java program.

- Use "`System.out.println`" method to identify problem.

```
public class Factors2 {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i < N; i++) {
         while (N % i == 0)
            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)

Debugging a Program: Performance Errors

Performance error: correct program but too slow.

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

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

Check if i is a factor.

As long as i is a factor, divide it out.

Too slow for large N (999,999,937)

Debugging a Program: Success

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

- Many fewer iterations of `for` loop.

```
public class Factors {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (long 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();
   }
}
```

Check if i is a factor.

As long as i is a factor, divide it out.

Special case: biggest factor occurs once.

Debugging a Program: Analysis

How big an integer can I factor?

```
% java Factors 168
2 2 2 3 7

% java Factors 6065102027
1009 2003 3001

% java Factors 9201111169755555703
9201111169755555703
```

3 minutes

largest factor

Digits	i <= N	i <= N / i	i * i <= N
3	instant	instant	instant
6	0.15 seconds	instant	instant
9	77 seconds	instant	instant
12	21 hours †	0.21 seconds	0.16 seconds
15	2.4 years †	4.5 seconds	2.7 seconds
18	2.4 millennia †	157 seconds	92 seconds

† estimated

Debugging a Program

Debug: cyclic process of editing, compiling, and fixing errors.

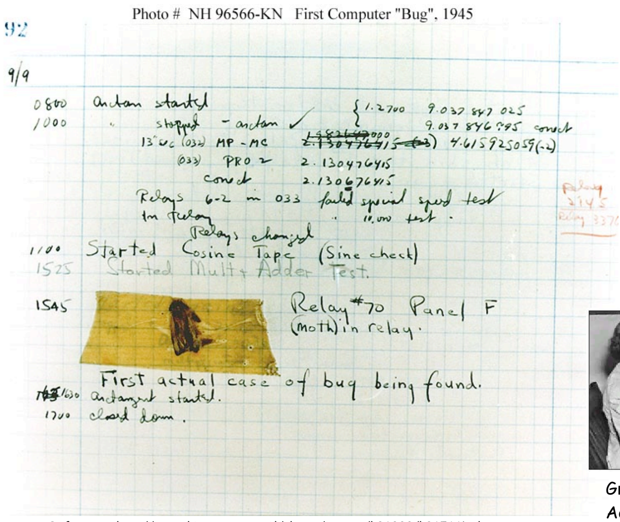
- Always a logical explanation.
- What would the machine do?
- Explain it to the teddy bear.

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*

Etymology and Entomology of Computer "Bug"



Reference: http://www.history.navy.mil/photos/images/h96000/h96566kc.htm

Grace Hopper
Admiral, US Navy

Flow Of Control Summary

Flow of control.

- Sequence of statements that are actually executed in a program.

Flow-Of-Control	*Description*	*Examples*
Straight-line programs	All statements are executed in the order given.	
Conditionals	Certain statements are executed depending on the values of certain variables.	`if` `if-else`

Loops	Certain statements are executed repeatedly until certain conditions are met.	`while`
		`for`
		`do-while`

Conditionals and loops.

- Simple, but powerful tools.
- Enables us to harness power of the computer.