INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

Conditionals and Loops



https://introcs.cs.rutgers.edu

Context: basic building blocks for programming

any program you might want to write

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!



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

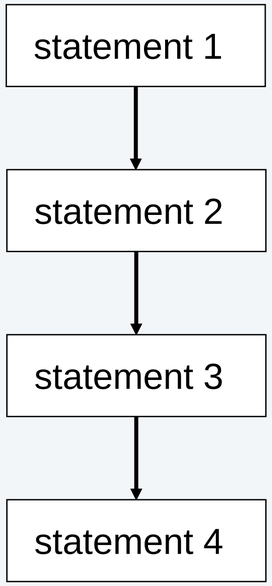
Conditionals & Loops

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

adopted and modified from Slides by Sedgewick and Wayne

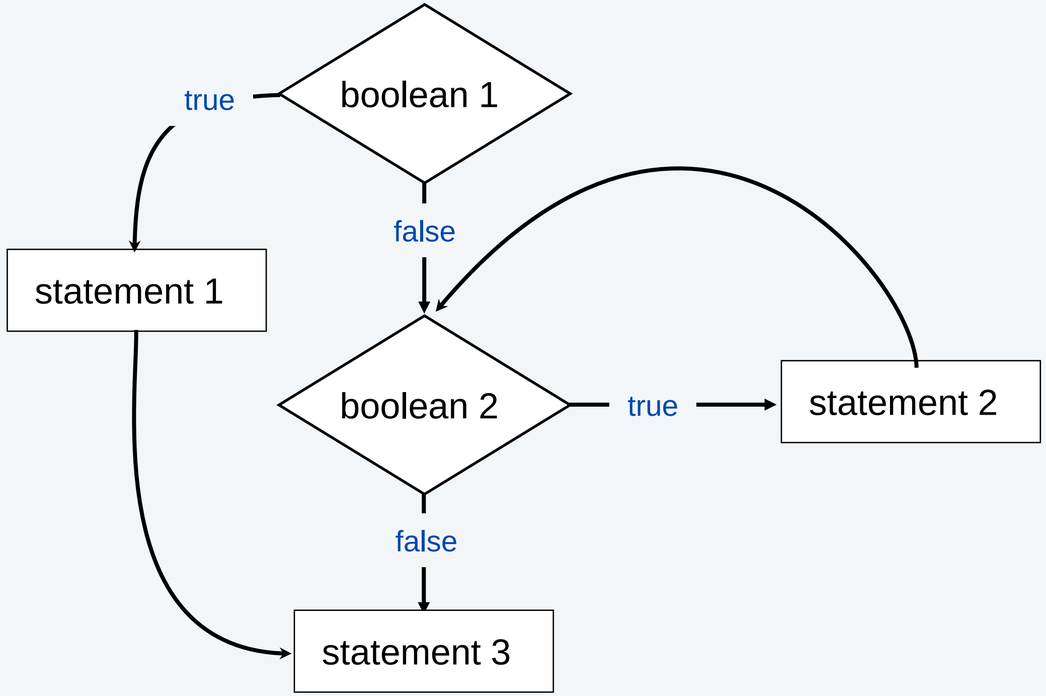
Control flow

- The sequence of statements that are 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

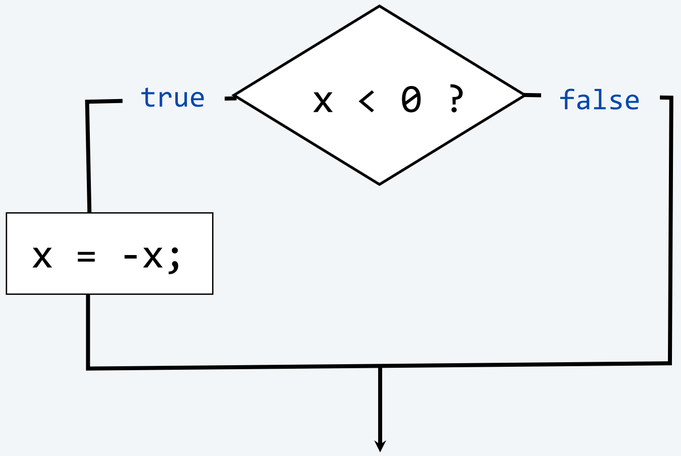
[this lecture]

The if-statement

Execute certain statements depending on the result of certain expressions.

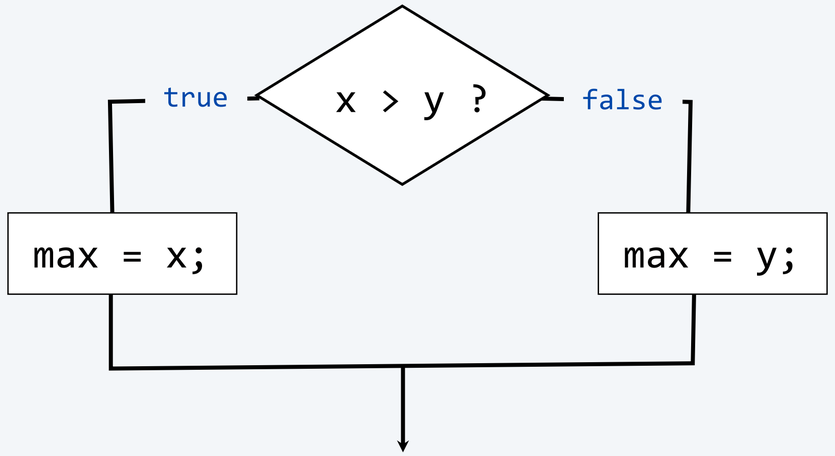
- Evaluate a boolean expression.
- If true, execute a statement.
- The else option: If false, execute a different statement.

Example: `if (x < 0) { x = -x; }`



Replaces x with 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

```
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
```



Anatomy of an if-statement

```
if ( conditional-expression )
{
    // if-block
} else {
    // (optional) else-block
}
```

Conditionally execute a block of code based on the value resulting from evaluating an expression

```
// What does this do?
int balance = 50;

if ( balance < 1000 )
{
    System.out.println("too little");
}
```

Example of if-statement

Q. What does this program do?

```
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;
        }
        System.out.println(a);
        System.out.println(b);
    }
}
```

Let's trace this

```
% java TwoSort 1234 99
99
1234
```

a	b	t
1234	99	
1234	99	1234
99	99	1234
99	1234	1234

```
% java TwoSort 99 1234
99
1234
```

a	b	t
99	1234	
99	1234	

Reads two integers from the command line, then prints them out in numerical order.

Pop quiz on if-statements

```
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]);









        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

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

```
% java ThreeSort 1234 99 1
1
99
1234

% java ThreeSort 99 1 1234
1
99
1234
```

Pop quiz on if-statements

```
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]);

        if (b < a) {
            int t = a;
            a = b;
            b = t;
        }
        if (c < a) {
            int t = a;
            a = c;
            c = t;
        }
        if (c < b) {
            int t = b;
            b = c;
            c = t;
        }

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

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

```
% java ThreeSort 1234 99 1
1
99
1234
```

a	b	c	t
1234	99	1	
99	1234	1	1234
1	1234	99	99
1	99	1234	1234

```
% java ThreeSort 99 1 1234
1
99
1234
```

a	b	c	t
99	1	1234	
1	99	1234	99

Example of if-statement use: error checks

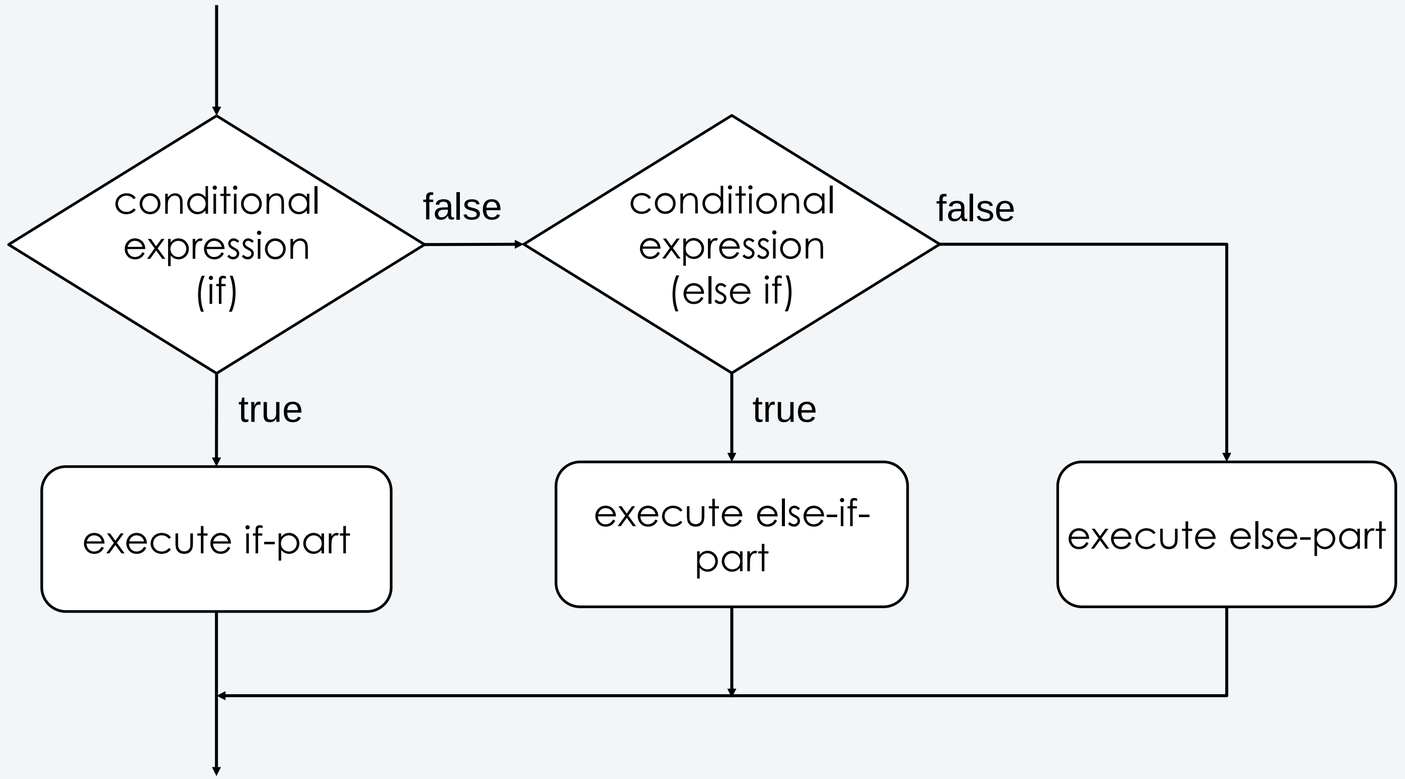
```
public class IntOps {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int sum  = a + b;
        int prod = a * b;
        System.out.println(a + " + " + b + " = " + sum);
        System.out.println(a + " *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

```
% java IntOps 5 2
5 + 2 = 7
5 * 2 = 10
5 / 2 = 2
5 % 2 = 1

% java IntOps 5 0
5 + 0 = 5
5 * 0 = 0
Division by zero
Division by zero
```

Good programming practice. Use conditionals to check for *and avoid* runtime errors.

`if/else if` Statement



if/else if Statement

```
if ( conditional-expression-1 ) {
    //statements;
} else if ( conditional-expression-2 ) {
    //statements;
} else {
    //statements;
}
```



```
// What does this do?
int balance = 1500;

if ( balance < 1000 ) {
    System.out.println("too little");
} else if ( balance > 2000 ) {
    System.out.println("too much");
} else {
    System.out.println("Proceed with your purchase");
}
```



Conditionals & Loops

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

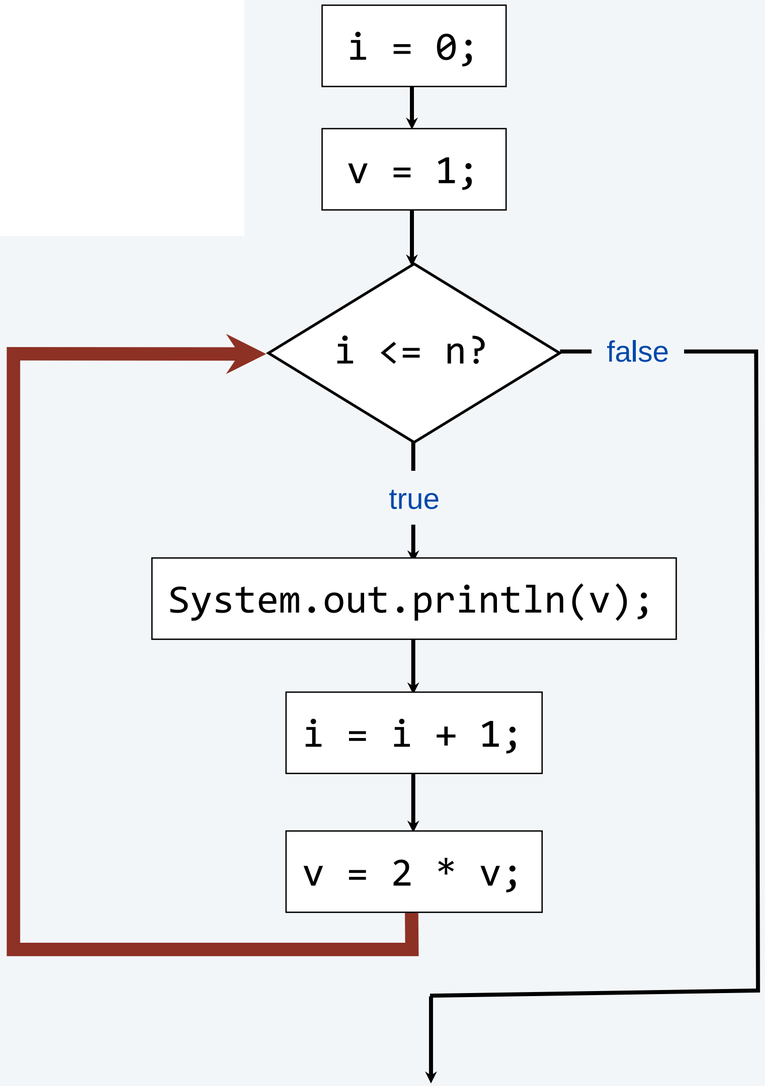
The while loop

Execute certain statements repeatedly until certain conditions are met.

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

Example:

```
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 .

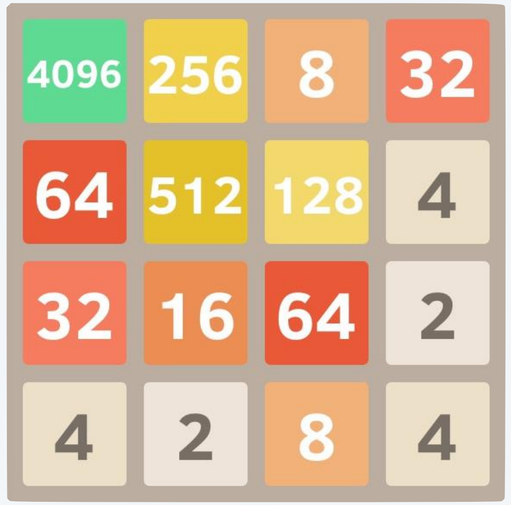
[stay tuned for a trace]

Example of while loop use: print powers of two

A trace is a table of variable values after each statement.

```
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;
        }
    }
}
```

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



Game: 2048

```
% java PowersOfTwo 6
1
2
4
8
16
32
64
```

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

powers of two : counting operations

Counting operations is a good way to know how well your program works.

```
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;
        }
    }
}
```

1 initialize operation

1 initialize operation

1 initialize operation

Loop condition checked for i= 0..n+1 (n+2 times)
Each operation in loop body is executed for i = 0..n (n+1 times) : 3 operations

1 print operation

1 increment/assignment operation

1 multiply/assignment operation

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

Total operations

$3 + (n+2) + 3(n+1)$

variable initialization

while-condition

loop operations

Anatomy of a while statement

```
while ( conditional-expression )
{
    // statements
}
```

Executes a block of statements *while* a *conditional-expression* continues to evaluate to `true`

Referred to as *Indefinite Iteration* because the number of times the code block repeated is not specified

```
double val = Math.random();     // Compute initial random number

while ( val < 0.9 ) {           // While random number is < 0.9
  System.out.println( val );
  val = Math.random();          // Compute another value to test
}
```

Pop quiz on while loops

Q. Anything wrong with the following code?

```
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;

    }
}
```

Pop quiz on while loops

Q. Anything wrong with the following code?

```
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;
        }
    }
}
```

A. Yes! Needs braces.

Q. What does it do (without the braces)?

A. Goes into an *infinite loop*.

```
% java PQwhile 6
1
1
1
1
1
1
1
1
1
1
1
1
1
```

KNOW HOW TO STOP AN ENDLESS LOOP ON YOUR COMPUTER! (control-C on Windows, Cmd-C on Mac)

← challenge: figure out how to stop it on your computer



Example of while loop use: implement Math.sqrt()

Goal. Implement square root function.

Start: $t^2 == c$
then: $t == c/t$

If you can find a t such that t = t/c, then t is $\sqrt{c}$

```
% java Sqrt 60481729.0
7777.0
% java Sqrt 2.0
1.4142136
```

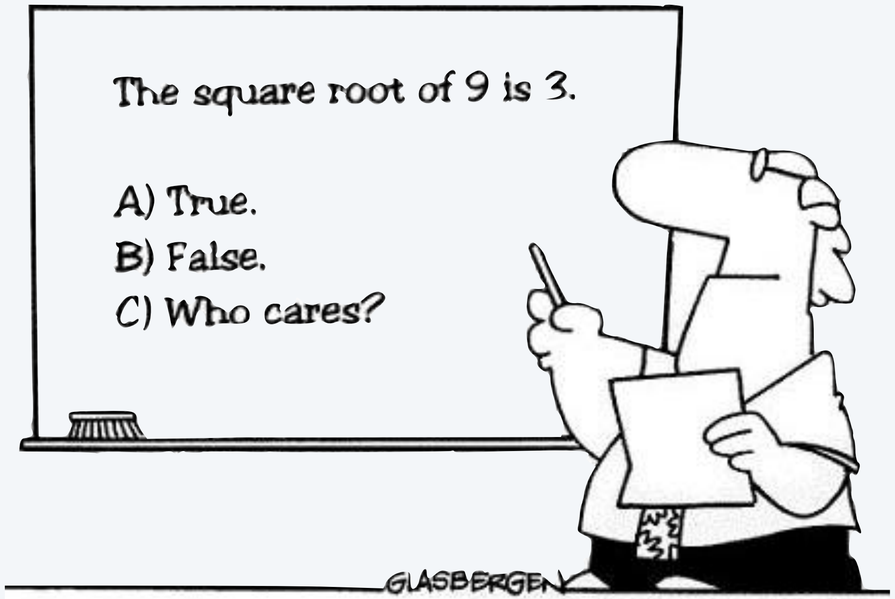
Newton-Raphson method to **compute $\sqrt{c}$**

- 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.

i	t_i	$2/t_i$	*average*
0	2	1	1.5
1	1.5	1.3333333	1.4166667
2	1.4166667	1.4117647	1.4142157
3	1.4142157	1.4142114	1.4142136
4	1.4142136	1.4142136	

computing the square root of 2 to seven places



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.



Isaac Newton
1642-1727

Scientists studied *computation* well before the onset of the *computer.*

```
public class Sqrt {
    public static void main(String[] args) {
        double EPS = 1E-15;
        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);
    }
}
```

error tolerance (15 places)

```
% java Sqrt 60481729.0
7777.0

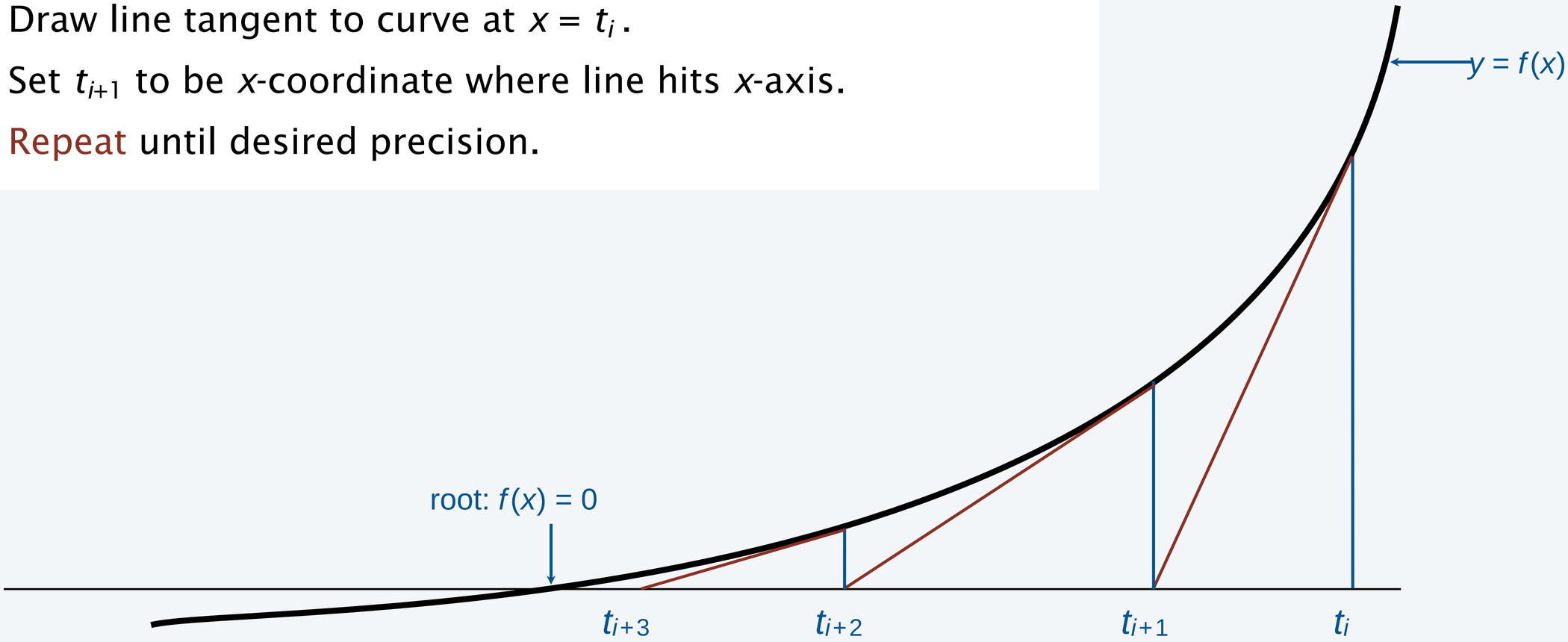
% java Sqrt 2.0
1.414213562373095
```

Newton-Raphson method

Explanation (some math omitted)

- Goal: find *root* of function $f(x)$ (value of x for which $f(x) = 0$).
- 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.

use $f(x) = x^2 - c$ for $\sqrt{c}$



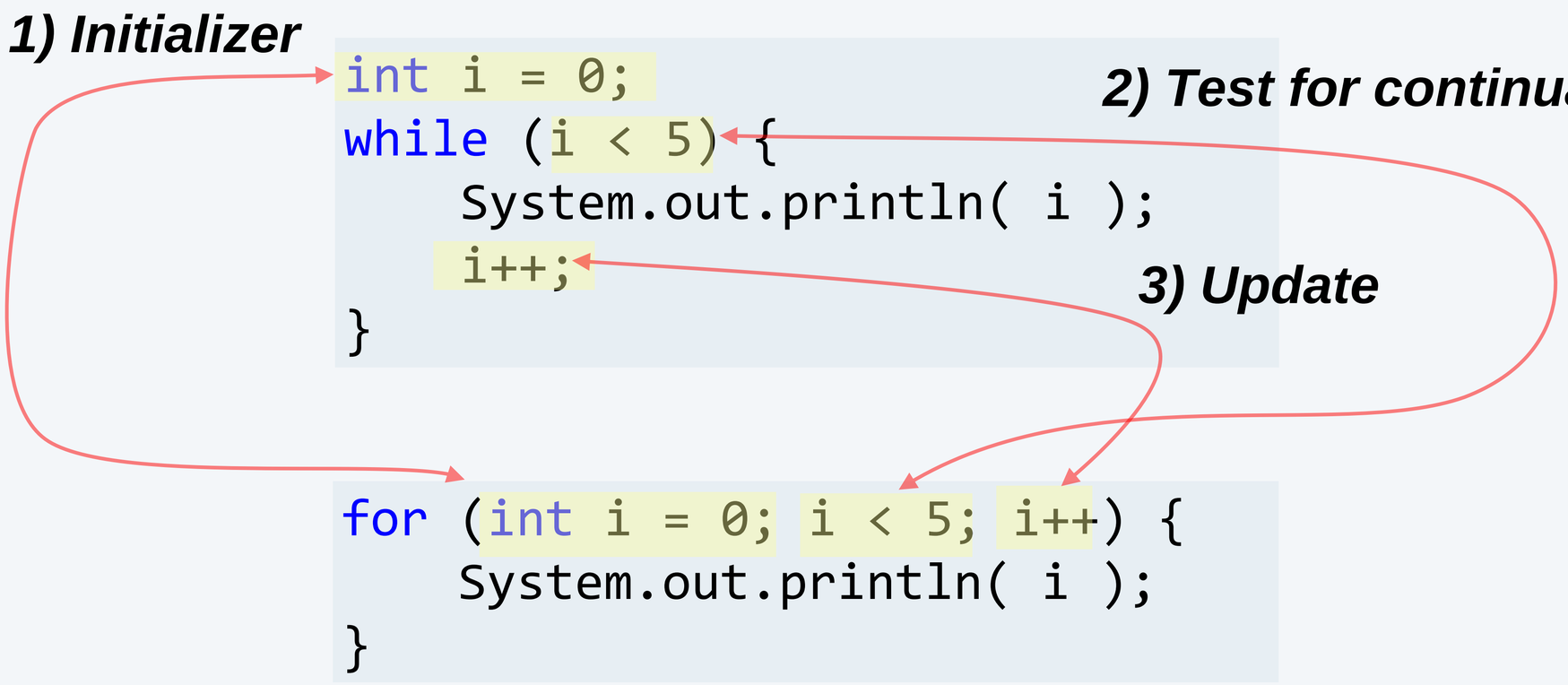


Conditionals & Loops

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

`for` statement vs. `while` statement: *Similarities*

The for-loop statement is an alternative to the **while-statement** plus **index counter pattern**



Note the repetition of multiple statements in two constructs

`for` statement : Another Way to Iterate

Three statements between parentheses of a for-statement
Separated by semi-colons

1. **Initializer**

   Executed once before the for-statement begins

2. **Continuation Test**

   A boolean expression

   *Evaluated **before** every loop iteration*

   Iterations continue if the expression evaluates to true

3. **Update**

   *Executed **after** each iteration*

```
for ( initializer; continuation-test; update )
{
    // Statements
}
```

`for` statement vs. `while` statement: *Differences*

The **scope of the two loop counters is different** if counter variable is declared in for-loop initializer

while-statement

- Loop counter declaration must be **outside the statement**
- Therefore, it exists in the **outer scope**
- **Loop counter and value persists** after the loop exits

```
int i = 0;
while (i < 5) {
    System.out.println( i );
    i++;
}
```

for-statement

- Loop counter declaration may be **within the statement** (but not required)
- In this case, it exists only **within the scope of the for-statement**
- …and the loop counter variable **does not exist in outer scope**

```
for (int i = 0; i < 5; i++) {
    System.out.println( i );
}
```

Scope of `for` loop variables

If a variable is declared in the `for`-loop header, its scope extends to the end of the `for` loop.

```
for (int x = 1; x <= n; x++){
        // x can be accessed here
}
// x is no longer defined.
```

You can declare before the for loop and initialize in the for loop.

```
int x;
for(x = 1; x <= n; x++){
       // x can be accessed here
}
// x is still in scope here
```

Examples of for loop use


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

Compute sum (1 + 2 + 3 + . . . + N)

sum	i
1	1
3	2
6	3
10	4

trace at end of loop for *N* = 4

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

Compute N! = 1 * 2 * 3 * . . . * N

product	i
1	1
2	2
6	3
24	4

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

Print a table of function values

k	$\frac{2\pi k}{N}$
0	0
1	1.57079632...
2	3.14159265...
3	4.71238898...
4	6.28318530...

```
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

v
2
4
8
16

trace at end of loop for *N* = 23

Example of for loop use: subdivisions of a ruler

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

- Initialize ruler to one space.
- For each value i from 1 to N:
  sandwich i between two copies of 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);
    }
}
```



1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

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 "

End-of-loop trace

```
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
```

Note: Small program can produce huge amount of output.

$2^{100} - 1$ integers in output (!)

Pop quiz on for loops

Q. What does the following program print?

```
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;
        }
    }
}
```

Pop quiz on for loops

Q. What does the following program print?

```
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;
        }
    }
}
```

Beginning-of-loop trace

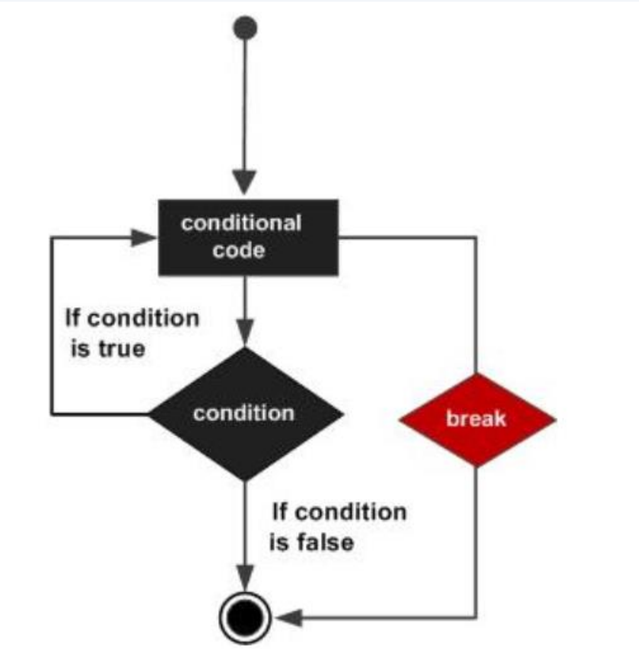
i	f	g
0	**0**	1
1	**1**	0
2	**1**	1
3	**2**	1
4	**3**	2
5	**5**	3
6	**8**	5
7	**13**	8
8	**21**	13
9	**34**	21
10	**55**	34

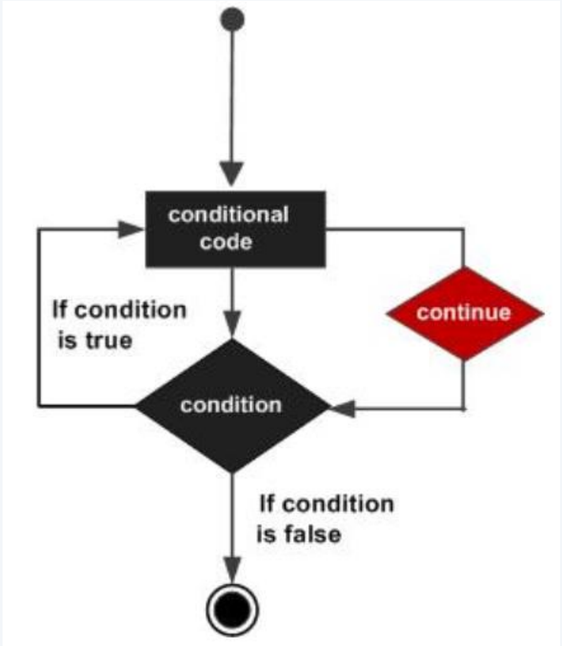
Printed values of f

Break and continue

break: exit the loop

continue: skip to end of the loop and continue to the next iteration






```
for (int i=0; i < n ; i++) {
      if ( i == 11)
            break;
      System.out.println(i);
}
```

break is executed when i is 11 which forces the end of the loop (exit the loop)

```
for (int i=0; i < n ; i++) {
      if ( i == 11)
            continue;
      System.out.println(i);
}
```

Loop skips printing 11

`break` Example

Loop 100 times, with a 1% chance of exiting the loop early

```
public class TestBreak
{
    public static void main(String[] args)
    {
        // Loop 100 times, with a 1% chance of exiting early
        for (int i = 0; i < 100; i++) {
            System.out.println(i);

            // 1% chance of exiting early
            if (Math.random() <= 0.01) {
                break;
            }
        }
        // Jumps here on break
    }
}
```

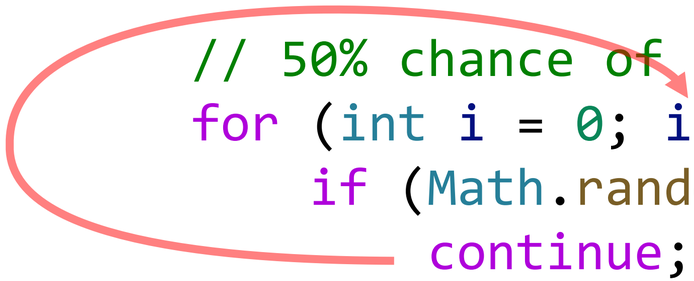
```
>java TestBreak
…
30
31
32
33
34
35
36
```

`continue` Example

Change all names more than four characters long to upper case.

```
public class TestContinue {
    public static void main(String[] args) {

        // 50% chance of skipping a number
        for (int i = 0; i < 100; i++) {      // Jumps back here on continue
            if (Math.random() < 0.5) {
                continue;
            }
            System.out.println(i);
        }
    }
}
```



```
>java TestContinue
0
2
3
6
7
8
11
12
```

Buggy code

Conditions

- What's wrong?
- What one change in the code below would cause in an infinite loop?

Example:

```
int i = 0;
while ( i > 1 ) {
    System.out.println(i);
    i++;
}
```

Buggy code

Conditions

- What's wrong?

Example:

```
int i = 10;
while ( i > 1 ) {
    System.out.println(i);
    i++;
}
```


The value of i is always greater than 1



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:

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

if-else statement within a while loop within a for loop

[Stay tuned for an explanation of this code.]

Example of nesting conditionals: Tax rate calculation

Goal. Given income, calculate proper tax rate.

income	rate
0 – $47,450	22%
$47,450 – $114,649	25%
$114,650 – $174,699	28%
$174,700 – $311,949	33%
$311,950 +	35%

```
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;
               }
         }
    }
```

if statement within an if statement

if statement within an if statement within an if statement

if statement within an if statement within an if statement within an if statement

Pop quiz on nested if statements

Q. Anything wrong with the following code?

```
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);
    }
}
```

Pop quiz on nested if statements

Q. Anything wrong with the following code?

```
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;
        else if (income < 114650)
            rate = 0.25;
        else if (income < 174700)
            rate = 0.28;
        else if (income < 311950)
            rate = 0.33;
        System.out.println(rate);
    }
}
```

Note. Braces are not needed in this case, but BE CAREFUL when nesting if-else statements because of potential ambiguity.

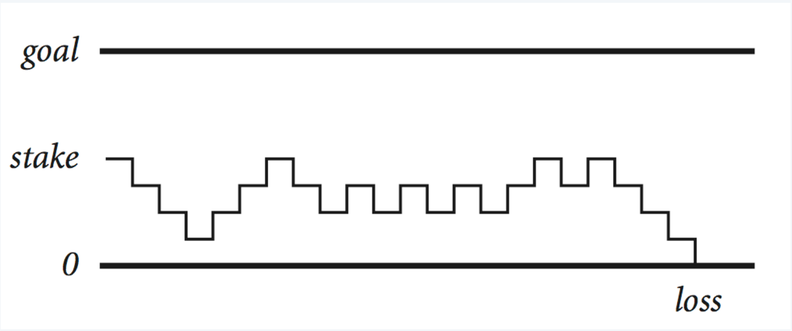
```
if (income < 311950) rate = 0.33;
else                 rate = 0.35;
```

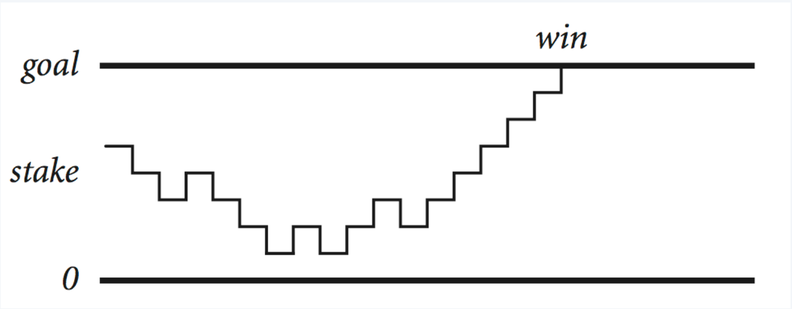
A. Yes! Need else clauses. Without them, code is equivalent to:

Gambler's ruin problem



*In statistics, gambler's ruin is the fact that **a gambler playing a game with negative expected value (mean) will eventually go broke**, regardless of their betting system.*





A gambler starts with $*stake* and places $1 fair bets.

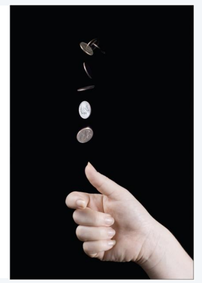
- Outcome 1 (loss): Gambler goes broke with $0.
- Outcome 2 (win): Gambler reaches $*goal*.

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

```
% java Gambler 5 25 1000
191 wins of 1000
```

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

for loop

while loop within a for loop

if statement within a while loop within a for loop



Conditionals & Loops

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

Debugging

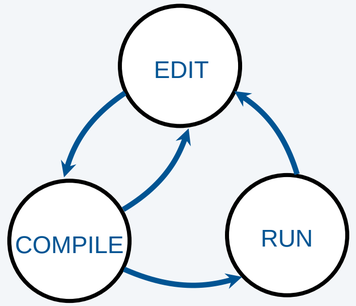
is 99% of program development in any programming language, *even for experts.*

Bug: A mistake in a program.

Debugging: The process of eliminating bugs.



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

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

Why is this impossible? Stay tuned.

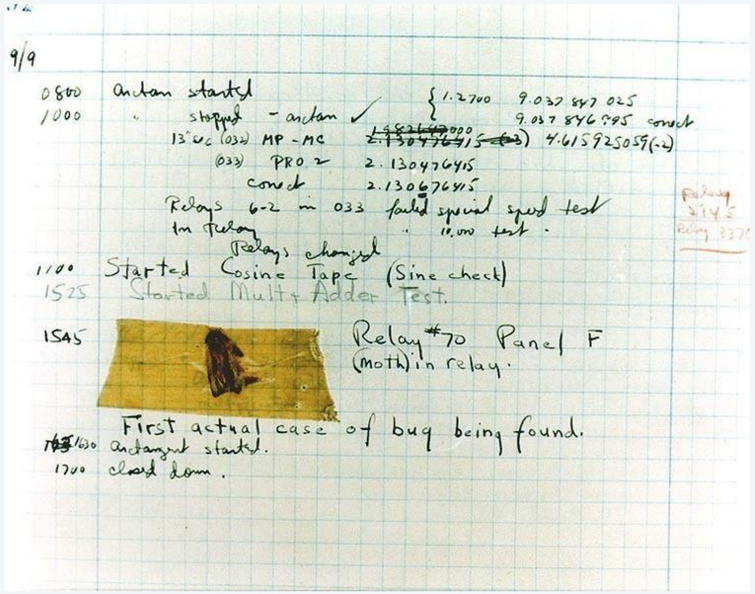
Bottom line: Programming is primarily a *process* of finding and fixing mistakes.

Debugging

is 99% of program development in any programming language, *even for experts.*



Grace Hopper











https://www.nationalgeographic.org/thisday/sep9/worlds-first-computer-bug/

Debugging

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

program structure	*no loops*	*n conditionals*	1 loop
number of possible execution sequences	1	2^n	no limit

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 *goto*s was controversial until Edsgar 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. ”

– Edsgar Dijkstra



Debugging a program: a running example

LO 4.1k

Problem: Factor a large integer *n*.

Application: Cryptography.

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

3,757,208 = 2 × 2 × 2 × 7 × 13 × 13 × 397

98 = 2 × 7× 7

17 = 17

11,111,111,111,111,111 = 2,071,723 × 5,363,222,357

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*.

```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0])
        for (i = 0; i < n; i++)
        {
            while (n % i == 0)
                System.out.print(i + " ")
                n = n / i

        }
    }
}
```







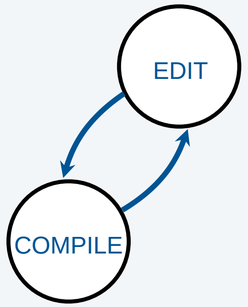


This program has bugs!

Debugging a program: syntax errors

Is your program a legal Java program?

- Java compiler can help you find out.
- Find the *first* compiler error (if any).
- Repeat.
- Result: An executable Factors.class file





Trying to tell a computer what to do

```
% javac Factors.java
Factors.java:5: ';' expected
        int n = Integer.parseInt(args[0])
                                         ^
...
```

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

```
% javac Factors.java
```



```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        for (int i = 0; i < n; i++)
        {
            while (n % i == 0) {
                System.out.print(i + " ");
                n = n / i;
            }
        }
    }
}
```

need to declare type of i

need terminating semicolons



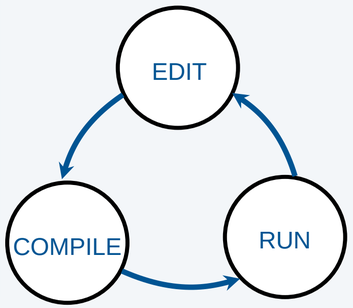


This legal program still has bugs!

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.
- Find the *first* runtime error (if any).
- Fix and repeat.





```
% javac Factors.java
% java Factors
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
        at Factors.main(Factors.java:5)
```

oops, need argument

you will see this message!

```
% 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 2 2 2 2 2 2
```

```
% java Factors 98
2 7 7%
```

98 = 2 × 7× 7 ✓

```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        for (int i = 2; i < n; i++)
        {
            while (n % i == 0) {
                System.out.print(i + " ");
                n = n / i;
            }
        }
    }
}
```

need to start at 2 since 0 and 1 are not factors

need braces





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 Factors 98
2 7 7%
```
need newline

```
% java Factors 5
```
??? no output

```
% java Factors 6
2
```
??? where's the 3?

```
% javac Factors.java
% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5
% java Factors 6
2
TRACE 2 3
```

AHA! Need to print out *n* (if it is not 1).

```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        for (int i = 2; i < n; i++)
        {
            while (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.



???
%$%@$#!!
forgot to recompile

```
% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5
% javac Factors.java
% java Factors 5
5
% java Factors 6
2 3
% java Factors 98
2 7 7
% java Factors 3757208
2 2 2 7 13 13 397
```

```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(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: where we left off

Problem: Factor a large integer *n*.

Application: Cryptography.



```
% java Factors 98
2 7 7%
```

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* .

```
% javac Factors.java
% java Factors 98
2 7 7
% java Factors 3757208
2 2 2 7 13 13 397
```

```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(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();
    }
}
```

performance analysis – counting operations

LO 4.1o

Q. How large an integer n, can i factor?

count the total operations if n = 24

```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(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();
    }
}
```

Important. Performance of the algorithm is related to number of operations

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.

change the *algorithm*: no need to check when $i > n/i$ since all smaller factors already checked

- 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* .

$i*i \le n$

```
% java Factors 11111111
11 73 101 137
% java Factors 11111111111
21649 513239
% java Factors 11111111111111
11 239 4649 909091
% java Factors 11111111111111111 2071723
```

might work, but way too slow

5363222357 ← immediate

```
public class Factors
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        for (int i = 2; i*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();
    }
}
```

Debugging a program: performance analysis

Q. How large an integer can I factor?

```
% java Factors 9201111169755555703
9201111169755555703
```

digits in largest factor	i < n	i*i <= n
3	instant	instant
6	instant	instant
9	77 seconds	instant
12	21 hours†	instant
15	2.4 years†	2.7 seconds
18	2.4 millenia†	92 seconds

† *estimated, using analytic number theory*



```
public class Factors
{
    public static void main(String[] args)
    {
        long n = Long.parseLong(args[0]);
        for (int i = 2; i*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();
    }
}
```

experts are still trying to develop better algorithms for this problem

Lesson. Performance matters!

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

Quiz. Generating test cases

Q. Consider the code that is intended to find the max of 3 integers. Generate test cases to assure that all possibilities are tested.

```
if (a > b)
    if (a > c)
       max = a;
    else
       max = c;
else
    if (b > c)
       max = b;
    else
       max = c;
```

Possible test cases. (not all of them)

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0

Important. To test your code properly, always generate enough test cases that can cover all cases

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/logic error

performance error



Telling a computer what to do when you know what you're doing

Image sources

http://playatlantic.com/sites/default/files/good%20looking%20girl%20on%20laptop_0.jpg

http://en.wikipedia.org/wiki/Edsger_W._Dijkstra#mediaviewer/File:Edsger_Wybe_Dijkstra.jpg

http://pixabay.com/en/hourglass-clock-timer-sand-time-152090/

Image sources

http://commons.wikimedia.org/wiki/File:Calculator_casio.jpg

http://en.wikipedia.org/wiki/File:Buzz-lightyear-toy-story-3-wallpaper.jpg

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2789164/#!po=30.0000 [181e306f1.jpg]

Image sources

http://www.sciencecartoonsplus.com

http://en.wikipedia.org/wiki/Isaac_Newton

http://www.onlinemathtutor.org/help/wp-content/uploads/math-cartoon-28112009.jpg

Image sources

http://pixabay.com/en/atlantic-city-ocean-holiday-316301/

http://en.wikipedia.org/wiki/Christiaan_Huygens#mediaviewer/File:Christiaan_Huygens.jpg



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

Conditionals & Loops

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

INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

Conditionals and Loops



https://introcs.cs.rutgers.edu