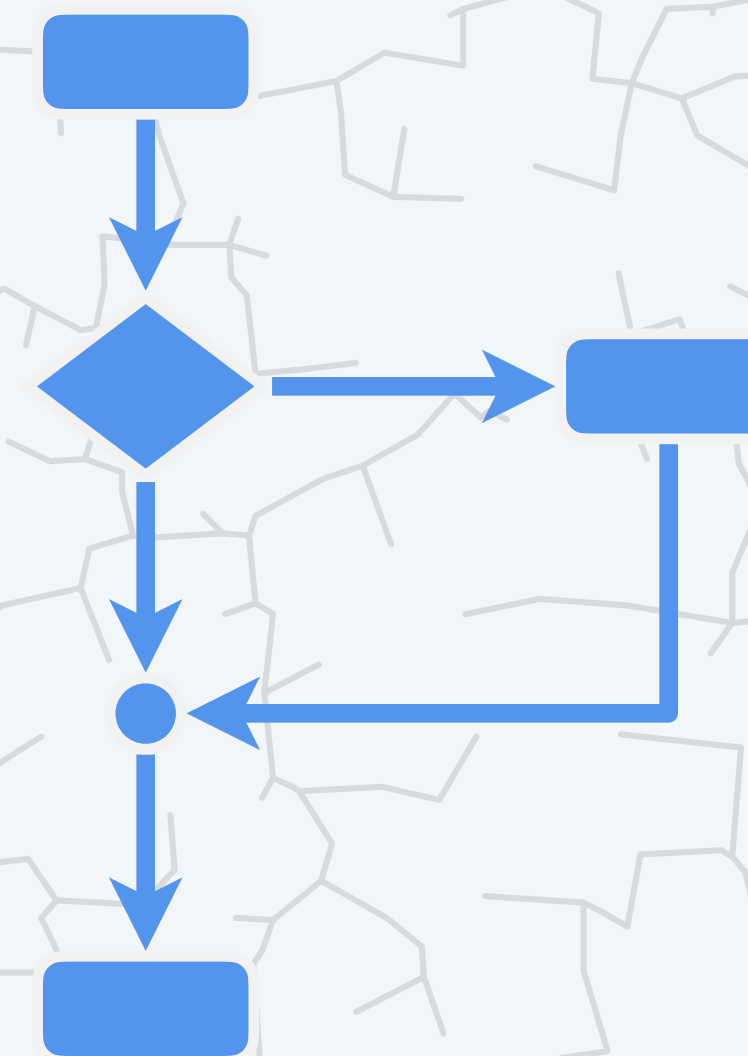


<https://introc.cs.princeton.edu>

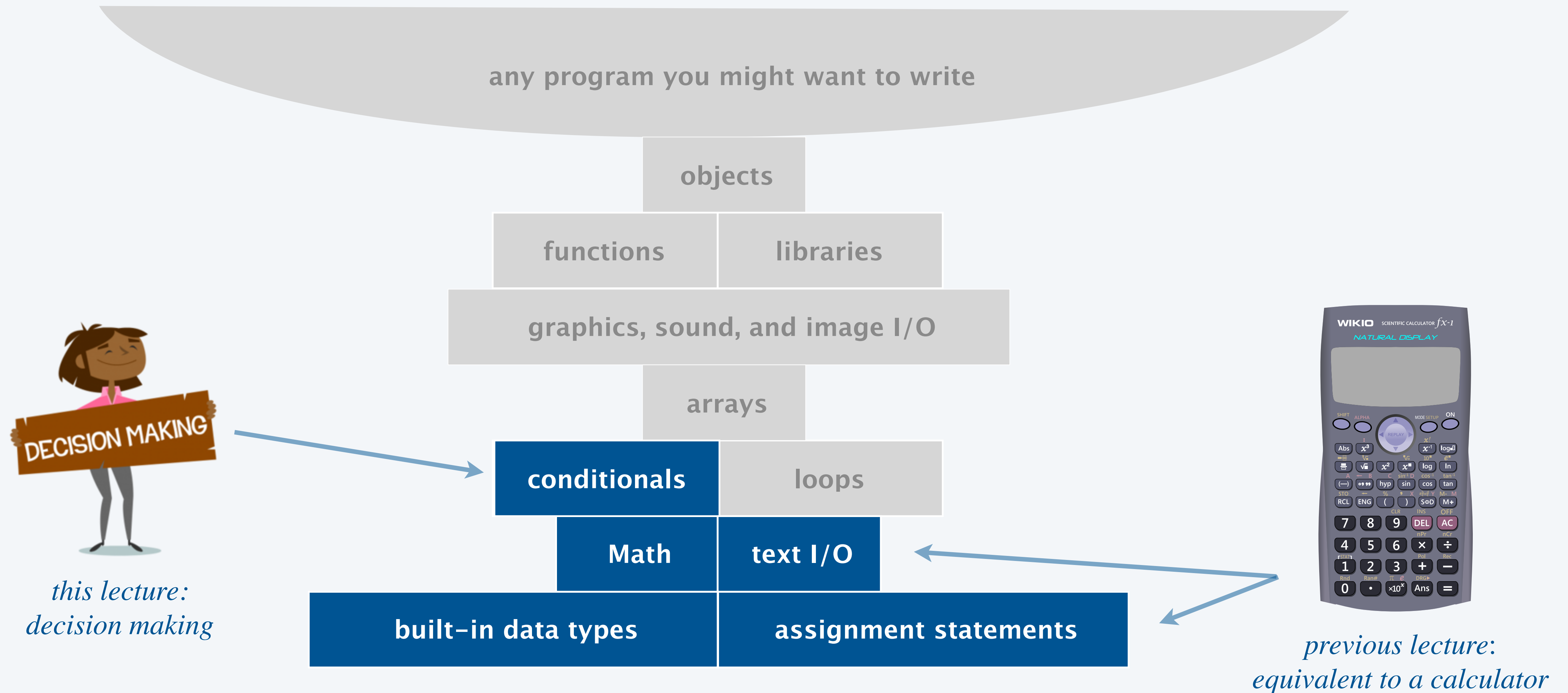
## 1.3 CONDITIONALS

---

- ▶ *booleans*
- ▶ *if statements*
- ▶ *if-else statements*
- ▶ *nested conditionals*
- ▶ *year-to-speech*



# Basic building blocks for programming

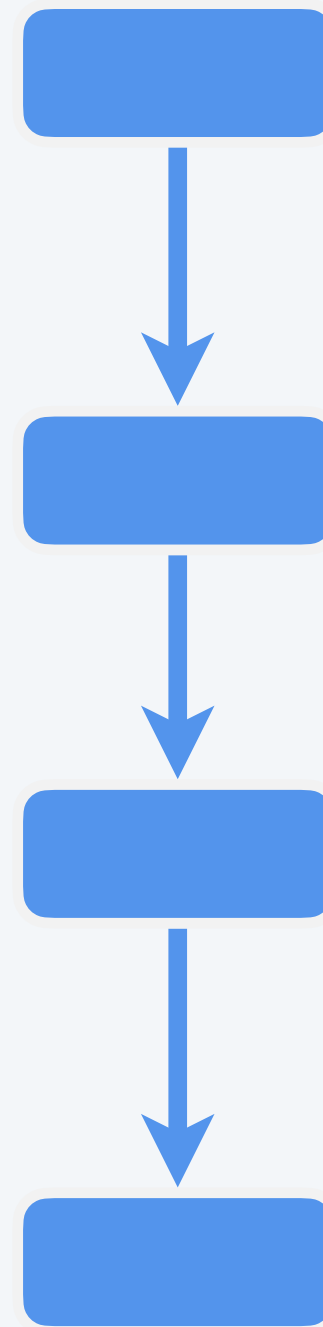


# Conditionals and loops

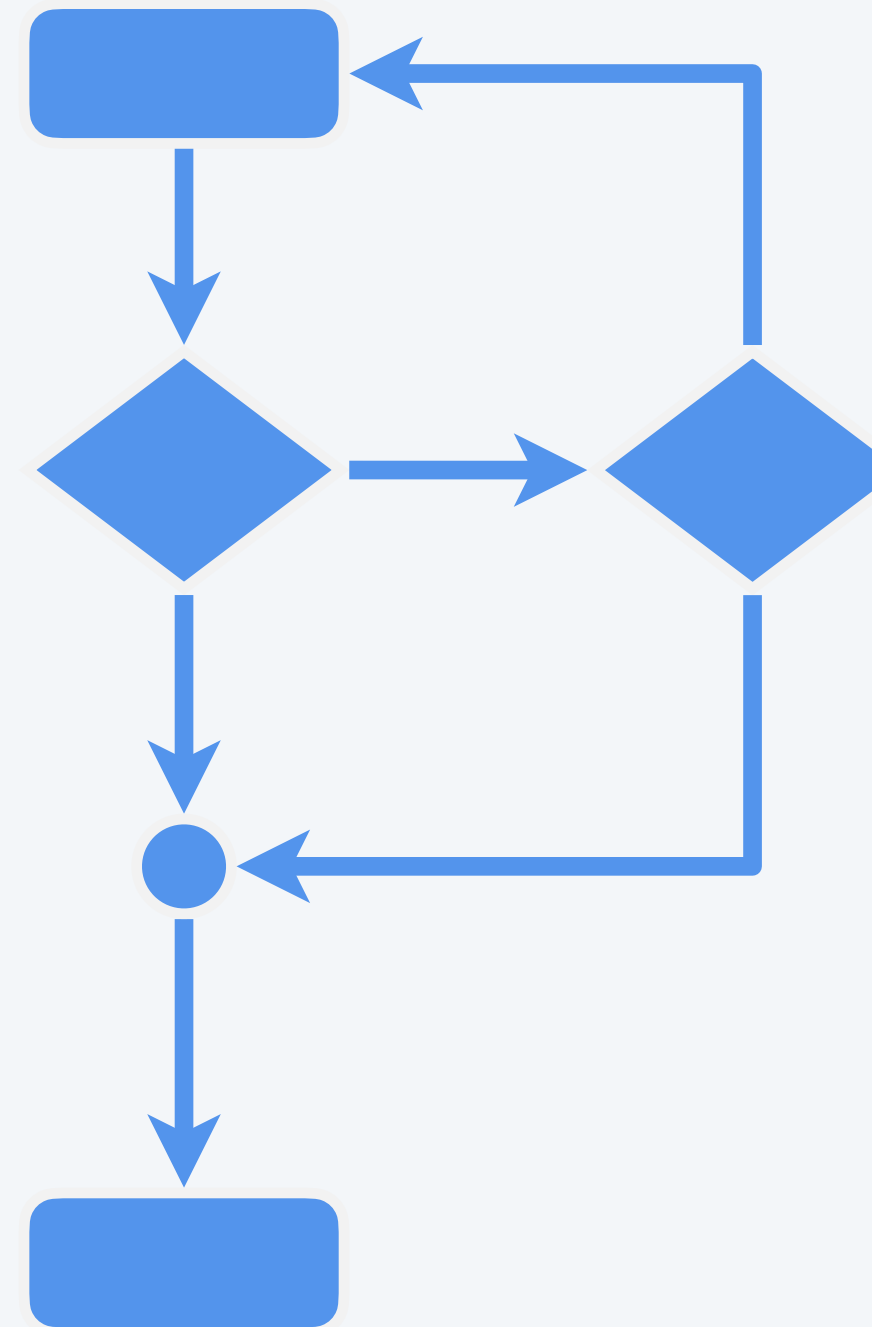
---

**Control flow.** The sequence of statements that are actually executed in a program.

**Conditionals and loops.** Enable us to choreograph control flow.

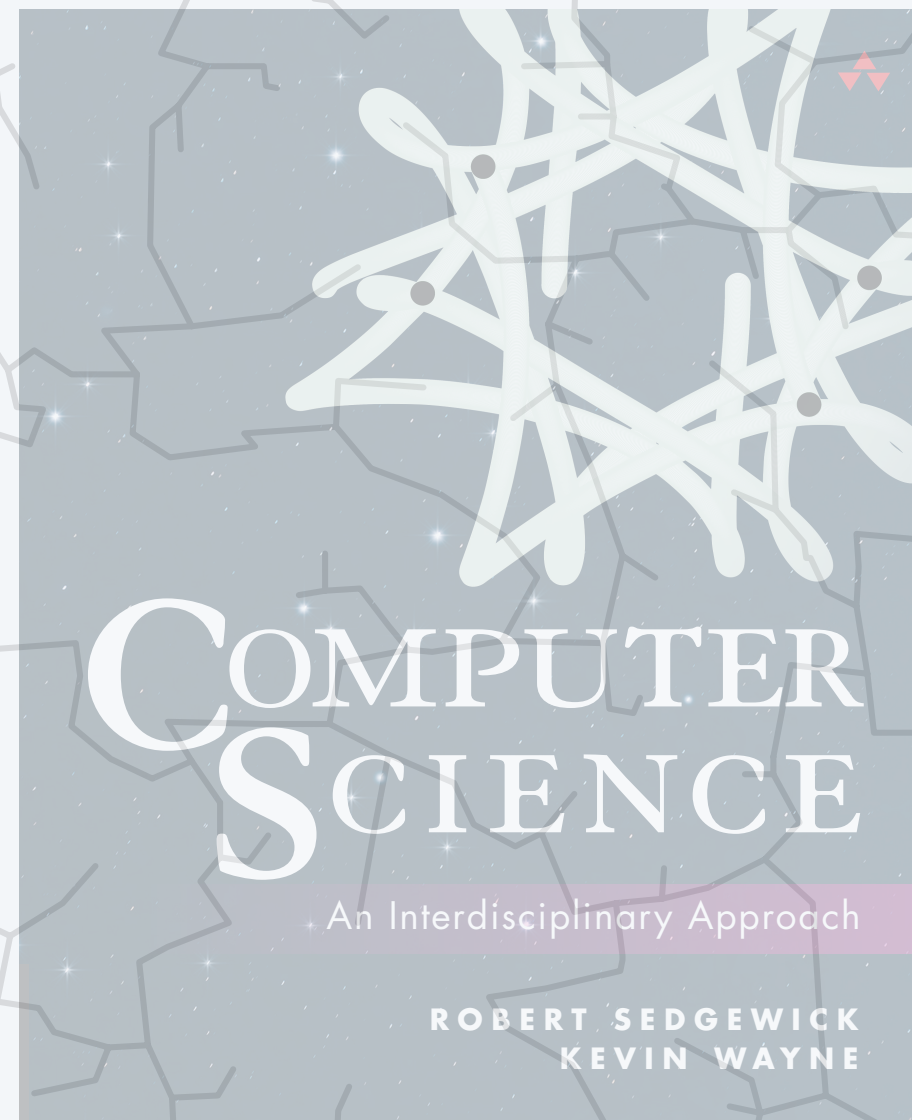


straight-line control flow  
(last lecture)



control flow with conditionals and loops  
(this week)





<https://introc.cs.princeton.edu>

## 1.3 CONDITIONALS

---

- ▶ *booleans*
- ▶ *if statements*
- ▶ *if-else statements*
- ▶ *nested conditionals*
- ▶ *year-to-speech*

## Built-in data types: review

---

A **data type (type)** is a set of values and a set of operations on those values.

type	set of values	example values	examples of operations
<i>String</i>	<i>sequences of characters</i>	"Hello, World" "COS 126 is fun!"	<i>concatenate</i>
<i>int</i>	<i>integers</i>	17 -12345	<i>add, subtract, multiply, divide, compare, equality</i>
<i>double</i>	<i>floating-point numbers</i>	2.5 -0.125	<i>add, subtract, multiply, divide, compare, equality</i>
<i>boolean</i>	<i>truth values</i>	true false	<i>and, or, not, equality</i>

**Java's built-in data types  
(that we use regularly in this course)**

# The *boolean* data type

Typical usage: decision making in a program. ← *with conditionals and loops*

values	<i>true and false</i>		
literals	true	false	
operations	<i>not</i>	<i>and</i>	<i>or</i>
operators	!	&&	← <i>logical operators</i>

expression	value
<code>!false</code>	true
<code>!true</code>	false

truth table for NOT

expression	value
<code>false &amp;&amp; false</code>	false
<code>false &amp;&amp; true</code>	false
<code>true &amp;&amp; false</code>	false
<code>true &amp;&amp; true</code>	true

truth table for AND

expression	value
<code>false    false</code>	false
<code>false    true</code>	true
<code>true    false</code>	true
<code>true    true</code>	true


truth table for OR





# Equality and comparison operators

---

Equality and comparison operators. To **compare** numeric values.

- Operands: two numeric expressions.  *can be literals, variable, or arbitrary expressions*
- Evaluates to: a value of type *boolean*.

operator	meaning	true	false
==	<i>equal</i>	2 == 2	2 == 3
!=	<i>not equal</i>	3 != 2	2 != 2
<	<i>less than</i>	2 < 13	13 < 2
<=	<i>less than or equal</i>	2 <= 2	3 <= 2
>	<i>greater than</i>	13 > 2	2 > 13
>=	<i>greater than or equal</i>	2 >= 2	2 >= 3

  *comparison operators applicable to int and double*

equality and comparison operators in Java



# Equality and comparison operators: examples

---

zero denominator?	<code>denominator == 0</code>	
non-negative discriminant?	<code>(b*b - 4.0*a*c) &gt;= 0.0</code>	
divisible by 60?	<code>(minutes % 60) == 0</code>	
RGB color is not black?	<code>(red &gt; 0)    (green &gt; 0)    (blue &gt; 0)</code>	<i>compound boolean expressions</i>
valid month?	<code>(month &gt;= 1) &amp;&amp; (month &lt;= 12)</code>	
invalid month?	<code>!((month &gt;= 1) &amp;&amp; (month &lt;= 12))</code>	
string equality	<code>args[0] == "Hello"</code>	<i>don't compare strings with == (this expression evaluates to false)</i>



Which of the following code fragments check whether month is between 1 and 12?

I.

```
1 <= month <= 12
```

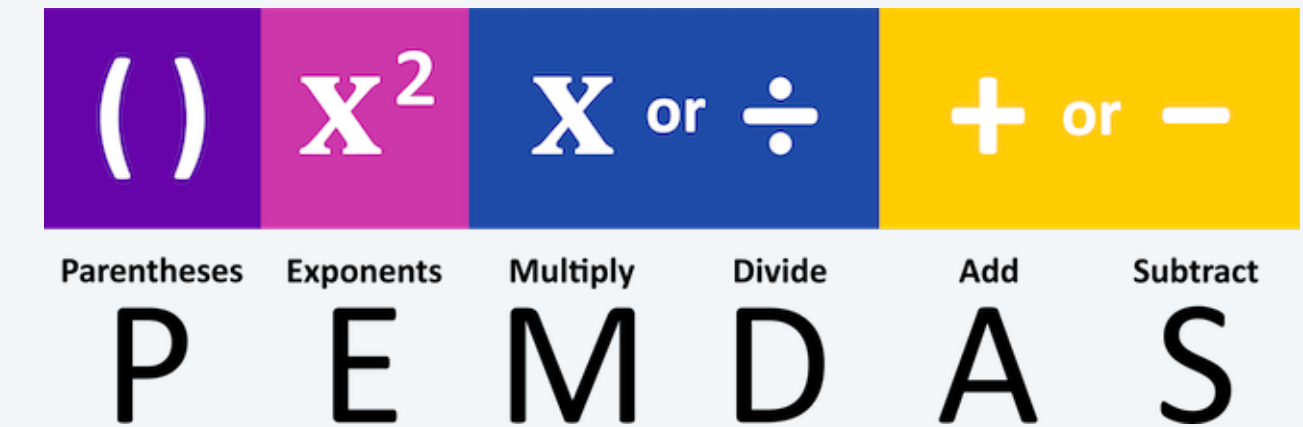
II.

```
month >= 1 && month <= 12
```

- A. I only.
- B. II only.
- C. I and II.
- D. Neither I nor II.

# Order of operations

**PEMDAS.** Grade-school rule for evaluating an arithmetic expression.

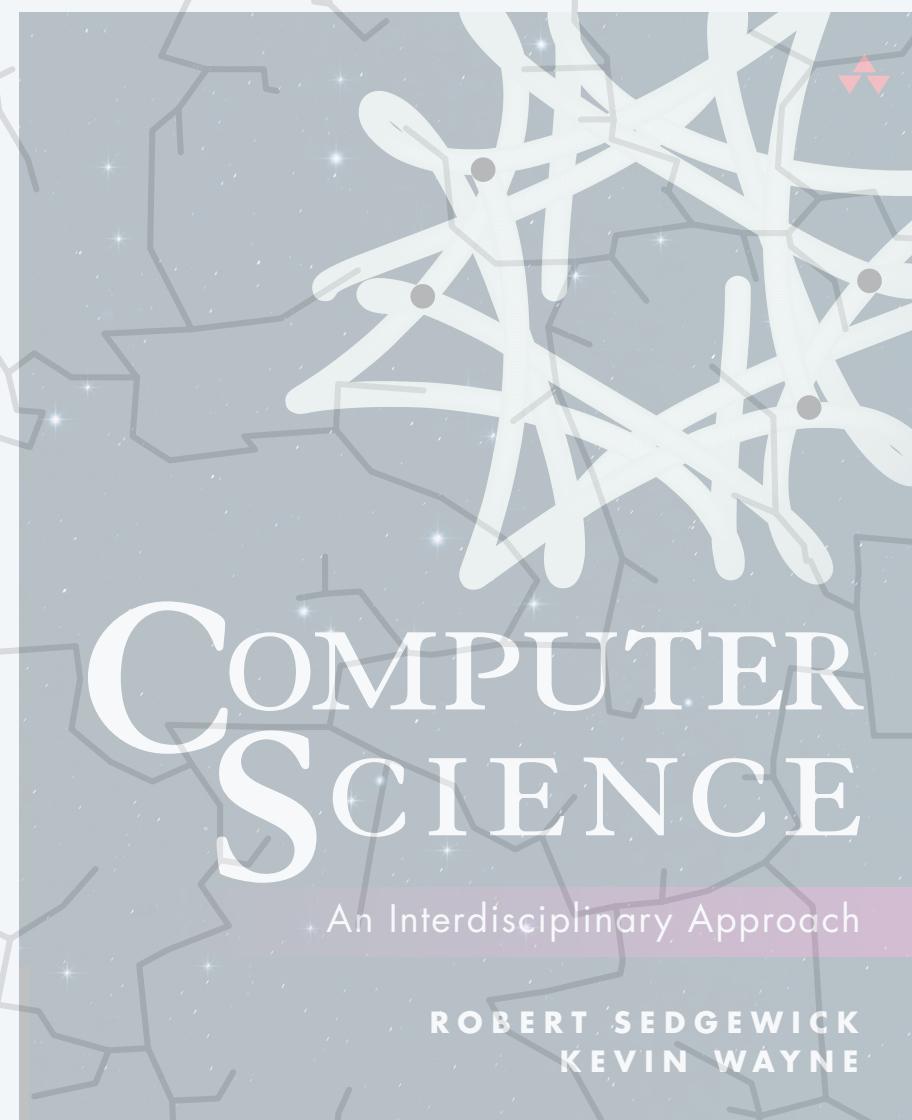


**Operator precedence.** Priority for grouping operands with operators in an expression.

**Operator associativity.** Rule for when two operators in an expression have same priority.

expression	equivalent to	value	remark
$10 - 2 * 3$	$10 - (2 * 3)$	4	<i>* has higher precedence than -</i>
$3 >= 1 \ \&\& \ 3 <= 12$	$(3 >= 1) \ \&\& \ (3 <= 12)$	true	<i>comparison operators have higher precedence than logical operators</i>
$3 - 5 - 2$	$(3 - 5) - 2$	-4	<i>left-to-right associative</i>
$3 * 5 / 2$	$(3 * 5) / 2$	7	<i>* and / have same precedence left-to-right associative</i>

← *Java has over 40 operators and 15 precedence levels (⇒ use parentheses for clarity)*



<https://introcs.cs.princeton.edu>

## 1.3 CONDITIONALS

---

- ▶ *if statements*
- ▶ *if-else statements*
- ▶ *nested conditionals*
- ▶ *year-to-speech*

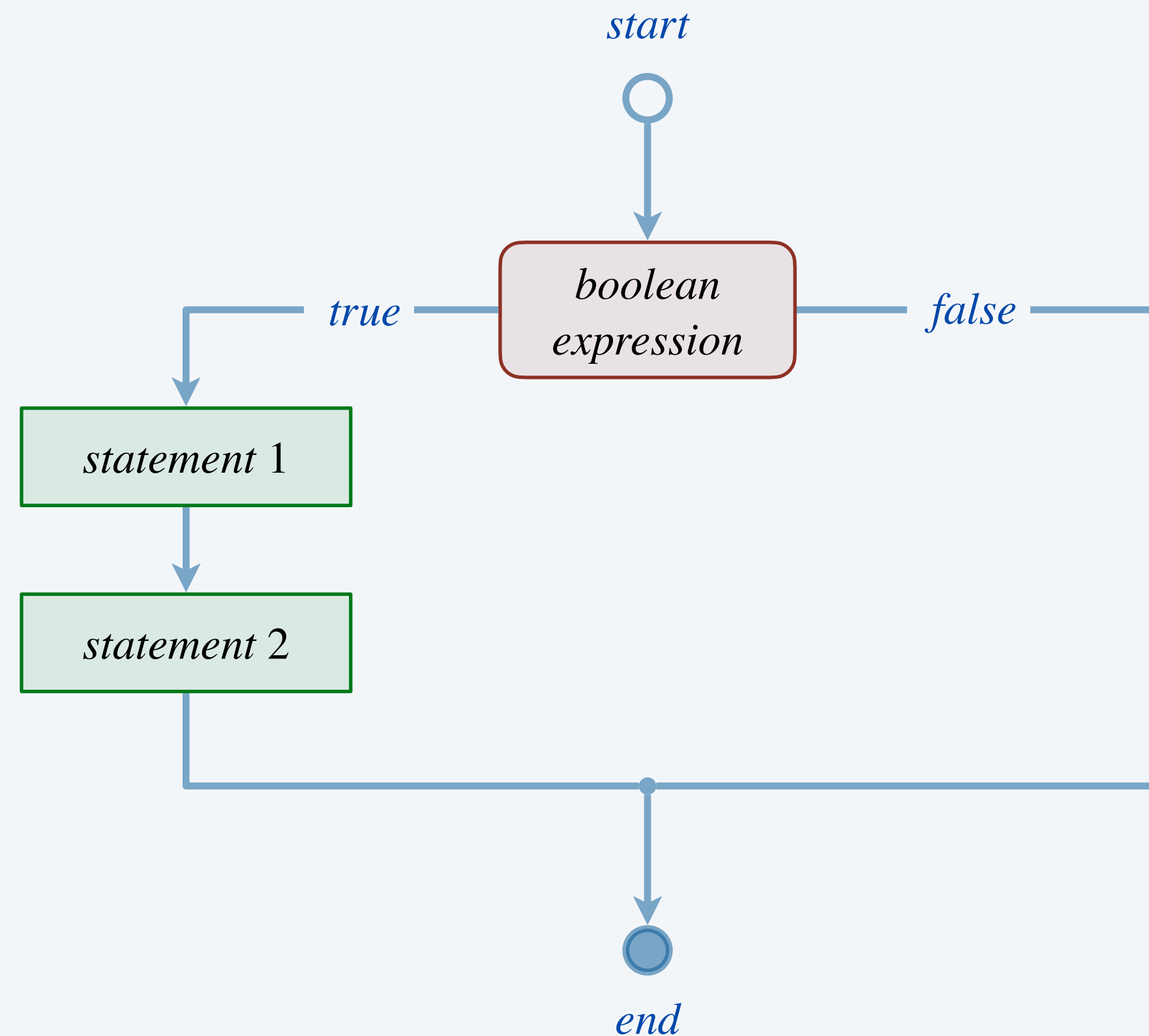
# The *if* statement

Execute certain statement(s) depending on the value of a boolean expression.

- Evaluate a boolean expression.
- If true, execute statements in **code block** delimited by curly braces.

```
if (<boolean expression>) {  
  <statement 1>  
  <statement 2>  
}
```

if statement template





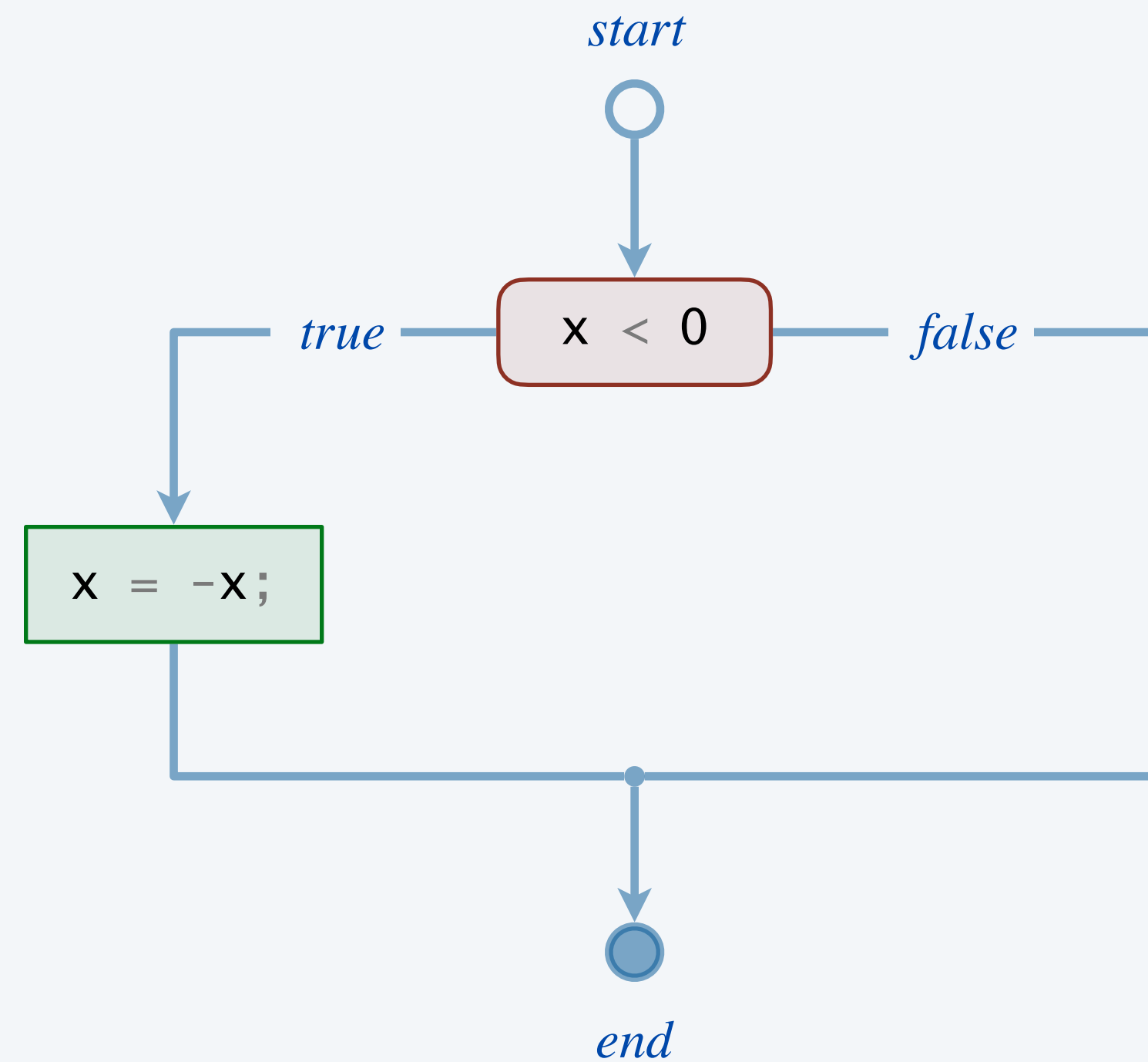
# The *if* statement

Execute certain statement(s) depending on the value of a boolean expression.

- Evaluate a boolean expression.
- If true, execute statements in **code block** delimited by curly braces.

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

replaces x with  
the absolute value of x



# Code blocks

A code block can contain a sequence of statements.

- Assignment statements.
- Declaration statements. ← *“local” variable accessible only within the block in which it is declared*
- Other *if* statements.
- ...

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

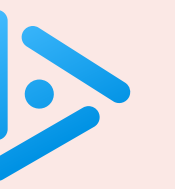
*code block consists of a sequence of statements (swap values in a and b)*

*temp not accessible here*

```
~/cos126/conditionals> java TwoSort 1234 126  
126  
1234  
  
~/cos126/conditionals> java TwoSort 126 1234  
126  
1234
```

## More examples of *if* statements

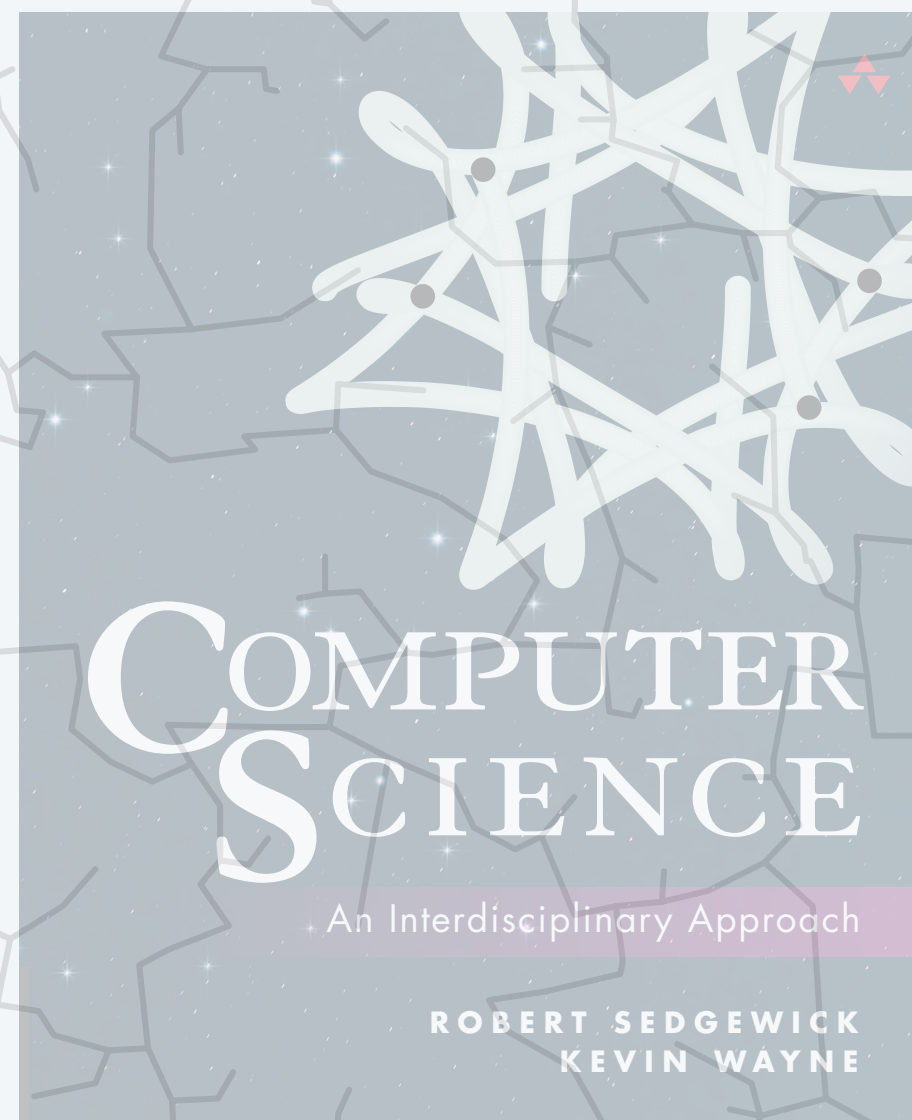
computation	for loop
<i>singular vs. plural</i>	<pre>String result = price + " dollar"; if (price != 1) {     result = result + "s"; }</pre>
<i>check if donor is ineligible to donate blood</i>	<pre>if ((age &lt; 16)    (weight &lt; 110)) {     System.out.println("ineligible"); }</pre> <p>← <i>compound boolean expression</i></p>
<i>time normalization</i>	<pre>if (minutes &gt;= 60) {     minutes = minutes - 60;     hours = hours + 1; }</pre> <p>← <i>multiple statements in body of if statement</i></p>
<i>maximum of three integers</i>	<pre>int max = a; if (b &gt; max) max = b; if (c &gt; max) max = c;</pre> <p>← <i>curly braces are optional since body of each if statement contains only one statement</i></p>



What does the following code fragment print?

```
int x = -126;  
if (x > 0); {  
    System.out.println("positive");  
};
```

- A. "positive"
- B. *nothing*
- C. *compile-time error*
- D. *run-time exception*



<https://introc.cs.princeton.edu>

## 1.3 CONDITIONALS

---

- ▶ *if statements*
- ▶ *if-else statements*
- ▶ *nested conditionals*
- ▶ *year-to-speech*



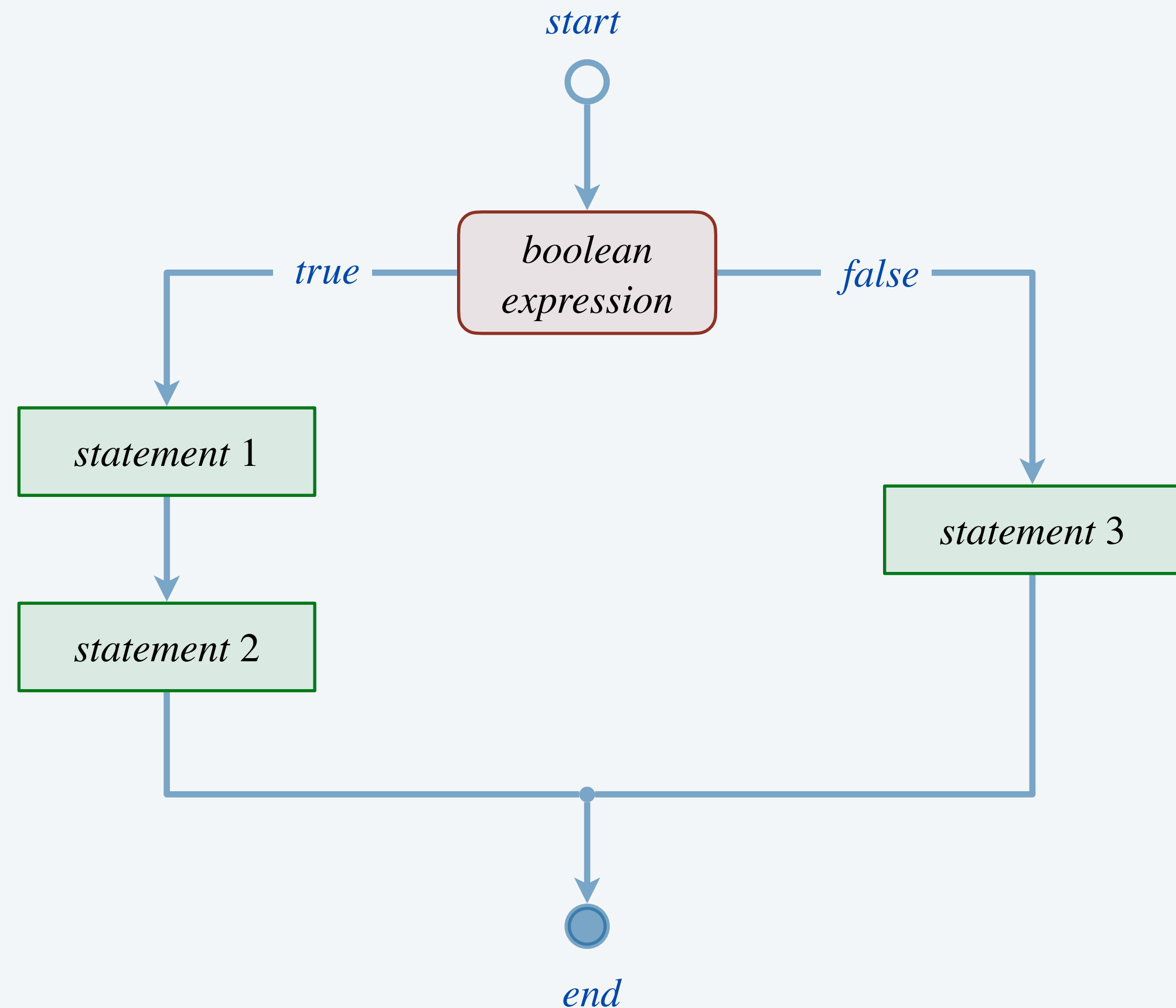
# The *if-else* statement

Execute certain statements depending on the value of a boolean expression.

- Evaluate a boolean expression.
- If true, execute some statements.
- Otherwise, execute different statements. ← *the else clause*

```
if (<boolean expression>) {  
    <statement 1>  
    <statement 2>  
}  
else {  
    <statement 3>  
}
```

if-else statement template



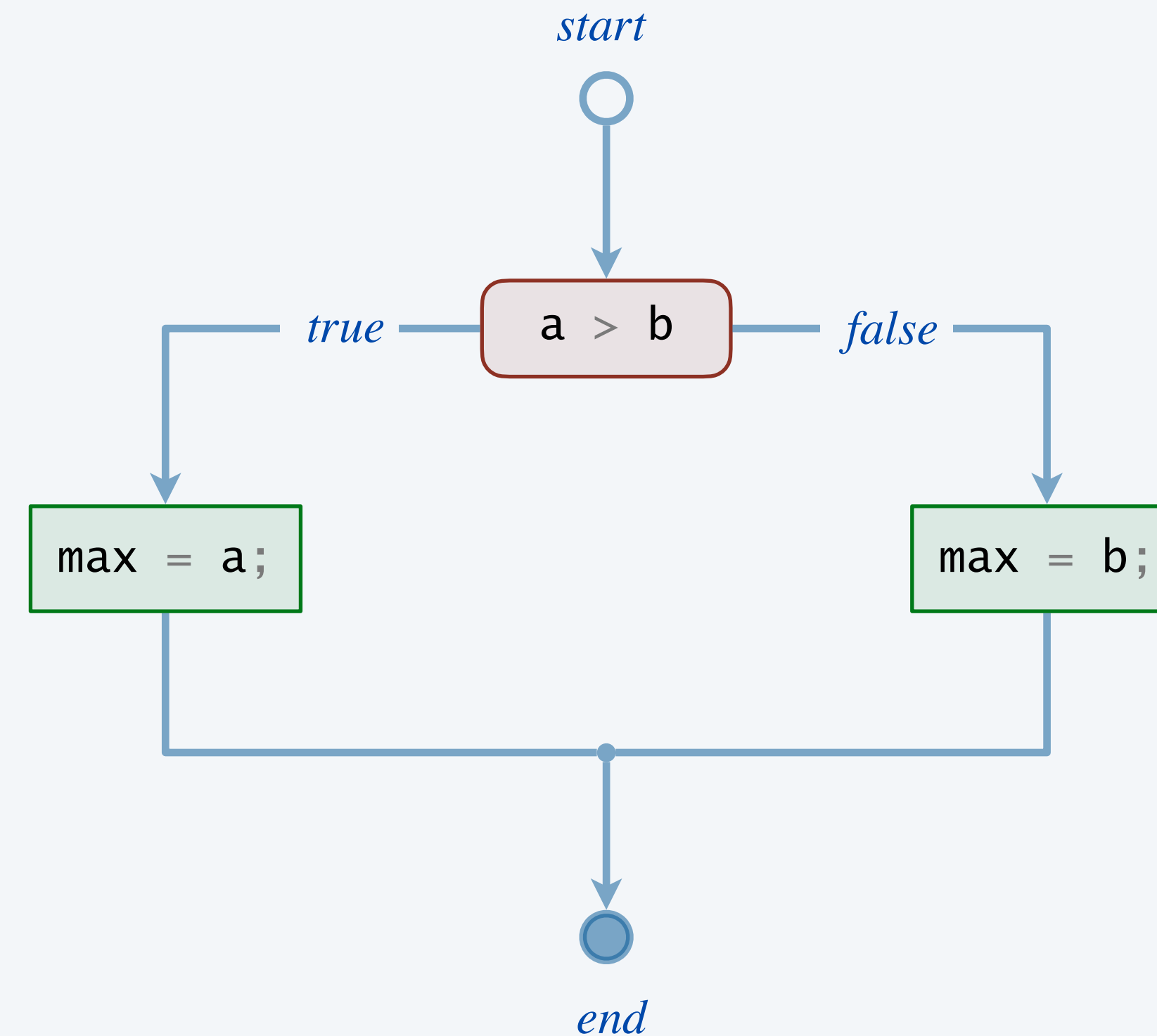
# The *if-else* statement

Execute certain statements depending on the value of a boolean expression.

- Evaluate a boolean expression.
- If true, execute some statements.
- Otherwise, execute different statements. ← *the else clause*

```
int max;  
if (a > b) {  
    max = a;  
}  
else {  
    max = b;  
}
```

sets max to the  
maximum of a and b



# Simulating a fair coin flip

---

**Goal.** Simulate a fair coin flip.



**Remark.** `Math.random()` returns a *double* value in the range  $[0, 1)$ .

```
public class CoinFlip {
    public static void main(String[] args) {
        double r = Math.random();

        if (r < 0.5) {
            System.out.println("Heads");
        }
        else {
            System.out.println("Tails");
        }
    }
}
```

```
~/cos126/conditionals> java CoinFlip
Heads

~/cos126/conditionals> java CoinFlip
Tails

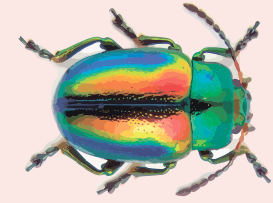
~/cos126/conditionals> java CoinFlip
Tails
```

## More examples of *if-else* statements

computation	if-else statement	
<i>simulating a gambler's fair bet</i>	<pre>double r = Math.random(); if (r &lt; 0.5) cash = cash + bet; else      cash = cash - bet;</pre>	← <i>if body consists of only one statement, so curly braces are optional</i>
<i>parity</i>	<pre>String parity; if (n % 2 == 0) parity = "even"; else          parity = "odd";</pre>	← <i>even: ..., -4, -2, 0, 2, 4, ...</i>
<i>integer remainder (with guard clause)</i>	<pre>if (denominator == 0) {     System.out.println("division by zero"); } else {     int remainder = numerator % denominator;     System.out.println("remainder = " + remainder); }</pre>	← <i>good style to include curly braces even when optional</i>



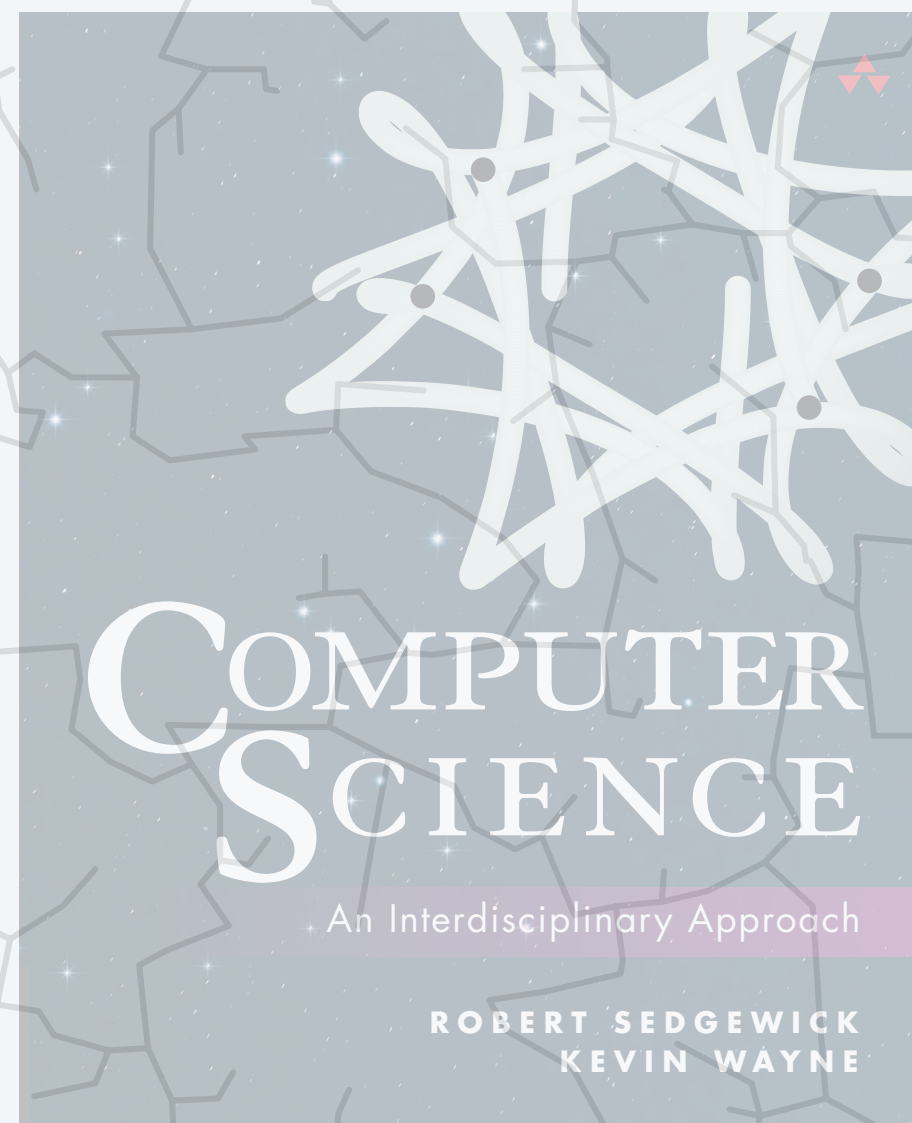
What does the following (buggy) code fragment print?



```
int x = -126;  
boolean isPositive = (x > 0);  
if (isPositive = true) System.out.println("positive");  
else System.out.println("not positive");
```

- A. "positive"
- B. "not positive"
- C. *nothing*
- D. *compile-time error*
- E. *run-time exception*





<https://introc.cs.princeton.edu>

## 1.3 CONDITIONALS

---

- ▶ *if statements*
- ▶ *if-else statements*
- ▶ *nested conditionals*
- ▶ *year-to-speech*



# Nesting conditionals: rock, paper, scissors

Three-way selection. Rock, paper, scissors.

```
public class RockPaperScissors {
    public static void main(String[] args) {
        int r = (int) (Math.random() * 3);

        if (r == 0) {
            System.out.println("Rock");
        }
        else {
            if (r == 1) {
                System.out.println("Paper");
            }
            else {
                System.out.println("Scissors");
            }
        }
    }
}
```

0, 1, or 2  
(see precept)

```
~/cos126/conditionals> java RockPaperScissors
Rock

~/cos126/conditionals> java RockPaperScissors
Scissors
```

*if-else statement nested  
within the else clause  
of an if statement*

# Nesting conditionals: types of triangle

**Triangle.** Given three angles of a triangle, is it invalid, acute, obtuse, right?

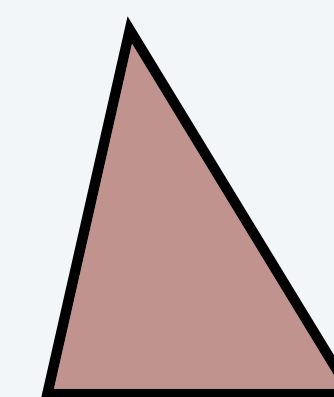
```
public class Triangle {  
    public static void main(String[] args) {  
        int a = Integer.parseInt(args[0]);  
        int b = Integer.parseInt(args[1]);  
        int c = Integer.parseInt(args[2]);  
  
        if (a <= 0 || b <= 0 || c <= 0 || (a + b + c != 180))  
            System.out.println("invalid");  
        else {  
            if (a < 90 && b < 90 && c < 90)  
                System.out.println("acute");  
            else {  
                if (a > 90 || b > 90 || c > 90)  
                    System.out.println("obtuse");  
                else  
                    System.out.println("right");  
            }  
        }  
    }  
}
```

*if statement nested within an if statement*

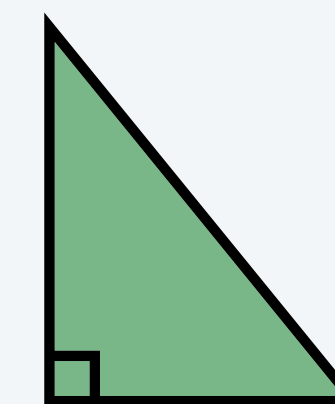
*if statement nested within an if statement within an if statement*

type	description
<i>invalid</i>	angles don't sum to 180°
<i>acute</i>	all angles less than 90°
<i>obtuse</i>	an angle greater than 90°
<i>right</i>	a 90° angle

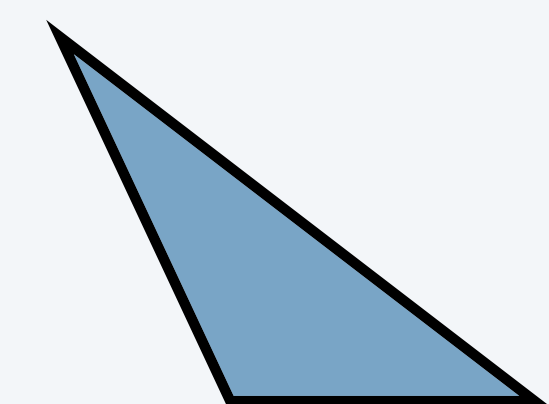
**mutually exclusive alternatives**



acute



right



obtuse

# Multiway selection shorthand

**Note.** Curly braces not needed here since each body consists of a single (compound) statement.

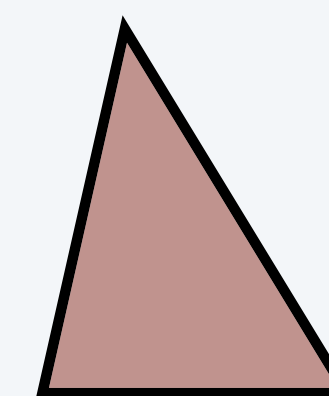
```
public class Triangle {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int c = Integer.parseInt(args[2]);

        if (a <= 0 || b <= 0 || c <= 0 || (a + b + c != 180))
            System.out.println("invalid");
        else if (a < 90 && b < 90 && c < 90)
            System.out.println("acute");
        else if (a > 90 || b > 90 || c > 90)
            System.out.println("obtuse");
        else
            System.out.println("right");
    }
}
```

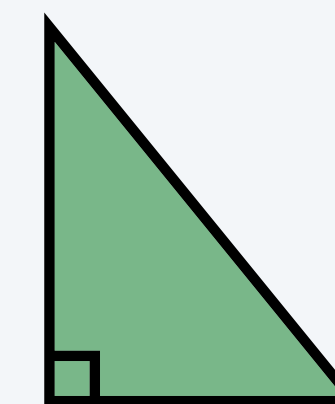
4 mutually  
exclusive  
alternatives

type	description
<i>invalid</i>	angles don't sum to 180°
<i>acute</i>	all angles less than 90°
<i>obtuse</i>	an angle greater than 90°
<i>right</i>	a 90° angle

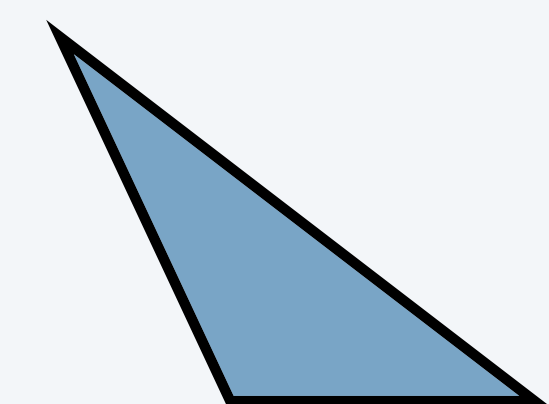
**mutually exclusive alternatives**



acute



right



obtuse

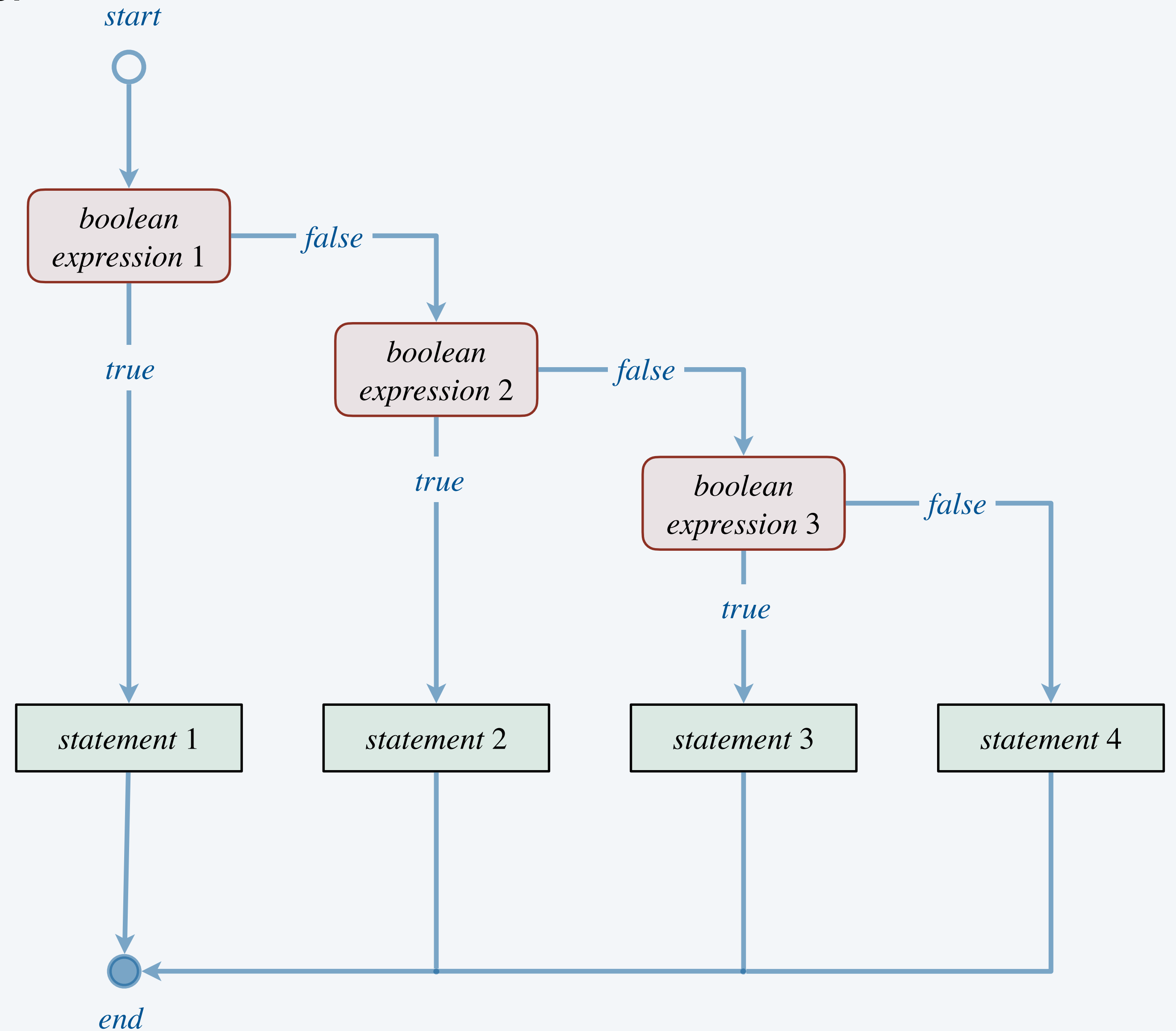


# A ladder of nested *if-else* statements

Multiway selection. Mutually exclusive alternatives.

```
if (<boolean expression 1>) {  
    <statement 1>  
}  
else if (<boolean expression 2>) {  
    <statement 2>  
}  
else if (<boolean expression 3>) {  
    <statement 3>  
}  
else {  
    <statement 4>  
}
```

if-else ladder template





# More examples of multiway selection

computation	nested if-else statements
<p data-bbox="559 559 902 607"><i>signum function</i></p> $\text{signum}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ +1 & \text{if } x > 0 \end{cases}$	<pre data-bbox="1212 615 1912 846">int signum; if (x &lt; 0) signum = -1; else if (x &gt; 0) signum = +1; else if      signum = 0;</pre> <p data-bbox="2525 746 3118 795">← 3 mutually exclusive alternatives</p>
<p data-bbox="369 1290 1086 1418"><i>Reynold's number</i> (ratio of inertial to viscous forces)</p>	<pre data-bbox="1212 1074 2292 1620">if (reynolds &lt;= 2000.0) {     System.out.println("laminar flow"); } else if (reynolds &gt;= 3500.0) {     System.out.println("turbulent flow"); } else {     System.out.println("transitional flow"); }</pre> <p data-bbox="2525 1258 3118 1307">← 3 mutually exclusive alternatives</p>



What will the following (buggy) code fragment print? Assume income is 100000.



- A. 0.22
- B. 0.25
- C. 0.28
- D. 0.33
- E. 0.35

```
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);
```

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

**marginal tax rate**

# Nested *if* statements

---

**Design principle.** Avoid unnecessary/gratuitous nesting of *if* statements.

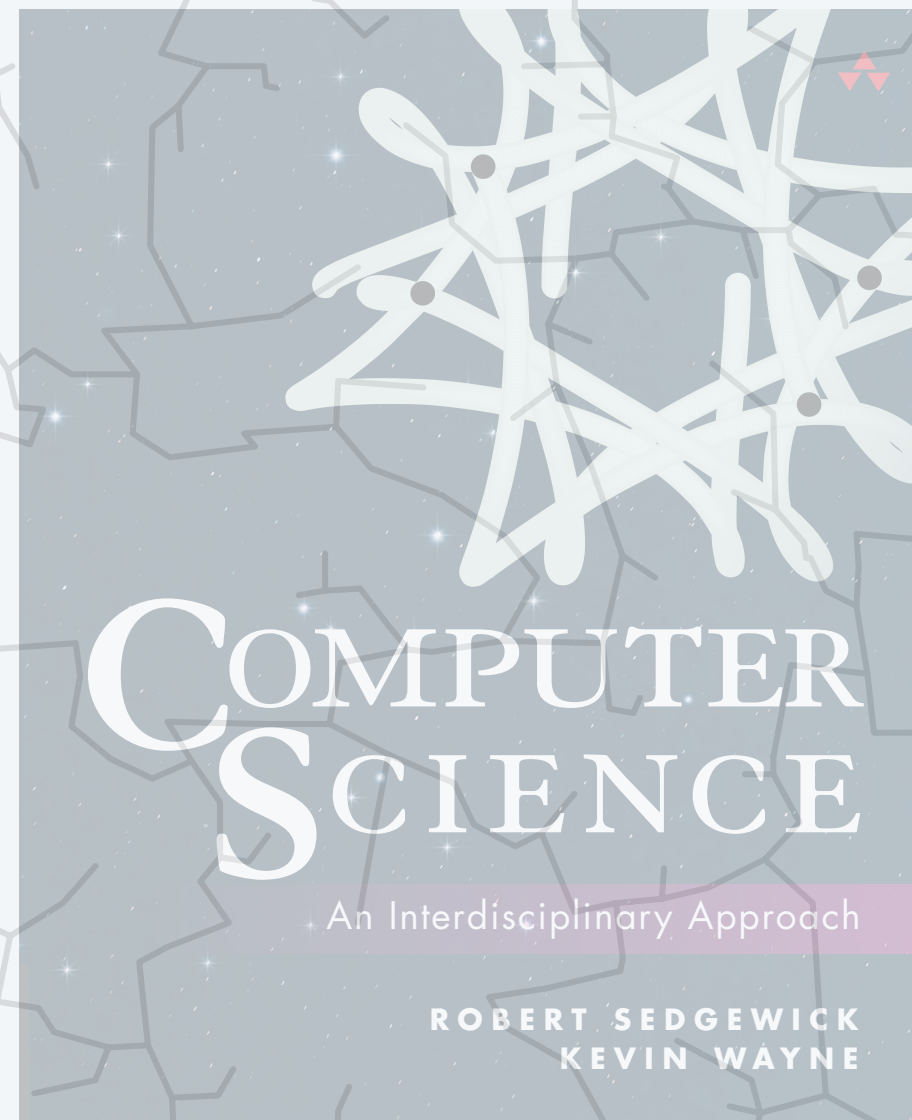
```
if (r == 0) {  
    if (g == 0) {  
        if (b == 0) {  
            System.out.println("black");  
        }  
    }  
}
```

**bad design (gratuitous nesting)**

```
if (r == 0 && g == 0 && b == 0) {  
    System.out.println("black");  
}
```

**easier to read and debug**





<https://introcs.cs.princeton.edu>

## 1.3 CONDITIONALS

---

- ▶ *if statements*
- ▶ *if-else statements*
- ▶ *nested conditionals*
- ▶ ***year-to-speech***



# Text-to-speech year

## Rules for speaking a year (1-9999) in English.

- Break up year into first-two and last-two digits; say each two-digit number.
- Special cases:
  - year ends in 000: say *thousand* for last three digits
  - year ends in 00 (but not 000): say *hundred* for last two digits
  - year ends in 01 to 09: say *oh* followed by single digit
  - year begins with 00: skip first two digits

year	spoken
2024	<i>twenty twenty-four</i>
1776	<i>seventeen seventy-six</i>
2000	<i>two thousand</i>
1700	<i>seventeen hundred</i>
1901	<i>nineteen oh one</i>
0026	<i>twenty-six</i>
12345	<i>invalid year</i>

**ENGLISH VOCABULARY** **The YEAR in English** Woodward ENGLISH

**Years**  
Years are normally divided into two parts.

**1984**  
*nineteen eighty-four*

**1066** *ten sixty-six*  
**1652** *sixteen fifty-two*  
**1941** *nineteen forty-one*  
**2017** *twenty seventeen*

When a year ends in a number between 01 and 09, then that last part is pronounced as the name of the letter O + number.  
**1709** *seventeen O nine*  
**1901** *nineteen O one*

When a year ends in 00 (e.g. 1600), then the year is said as the digits before 00, and then hundred.  
**1300** *thirteen hundred*  
**1800** *eighteen hundred*

**2000 - 2010**  
For the year 2000 you say (the year) **two thousand**.  
For the years 2001 to 2010, we normally say **two thousand and + number**.  
**2001** *two thousand and one*  
**2005** *two thousand and five*  
**2008** *two thousand and eight*

**After 2010**  
For the first years after 2010, you may hear two different versions.  
**2012** *two thousand and twelve*  
**2012** *twenty twelve*  
They are both used and correct. Now, we continue to say the year divided into two parts as before.

[www.grammar.cl](http://www.grammar.cl) [www.woodwardenglish.com](http://www.woodwardenglish.com) [www.vocabulary.cl](http://www.vocabulary.cl)



**Domain-specific synthesis.** Concatenate pre-recorded words to form desired output.



speaking the year 1901

word	audio file
1–99	1.wav, 2.wav, 3.wav, ...
<i>hundred</i>	hundred.wav
<i>thousand</i>	thousand.wav
<i>oh</i>	oh.wav
	<b>vocabulary</b>

## Applications.

- Talking clocks.
- Train schedule announcements.
- Interactive telephone voice response systems.

**Note.** Limited to words in vocabulary.





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

```
        int year = Integer.parseInt(args[0]);  
        int firstTwoDigits = year / 100;  
        int lastTwoDigits = year % 100;
```

← *assumes year is between 1 and 9999*

← *parse first and last two digits of year*

```
        if (year % 1000 == 0) {  
            int firstDigit = year / 1000;  
            StdAudio.play(firstDigit + ".wav");  
            StdAudio.play("thousand.wav");  
        }
```

← *special case for years ending in 000*

```
    else {
```

```
        if (firstTwoDigits > 0)  
            StdAudio.play(firstTwoDigits + ".wav");
```

← *say first two digits (unless 00)*

```
        if (lastTwoDigits == 0)  
            StdAudio.play("hundred.wav");
```

← *special case for years ending in 00 (but not 000)*

```
        else {  
            if (lastTwoDigits < 10)  
                StdAudio.play("oh.wav");
```

← *special case for years ending in 01 to 09*

```
            StdAudio.play(lastTwoDigits + ".wav");
```

← *say last two digits*

```
        }
```

```
    }
```

```
}
```

```
}
```



**Principle.** Supply inputs that activate all possible execution paths through program. ← *so that all code gets tested*



```
~/cos126/conditionals> java-introcs SayYear 2024 ← typical case
🔊 [speaks "twenty twenty-four"]

~/cos126/conditionals> java-introcs SayYear 1776 ← typical case
🔊 [speaks "seventeen seventy-six"]

~/cos126/conditionals> java-introcs SayYear 2000 ← year ends in 01 to 09
🔊 [speaks "two thousand"]

~/cos126/conditionals> java-introcs SayYear 1700 ← year ends in 000
🔊 [speaks "seventeen hundred"]

~/cos126/conditionals> java-introcs SayYear 1901 ← year ends in 00 (but not 000)
🔊 [speaks "nineteen oh one"]

~/cos126/conditionals> java-introcs SayYear 26 ← year begins with 00
🔊 [speaks "twenty-six"]
```

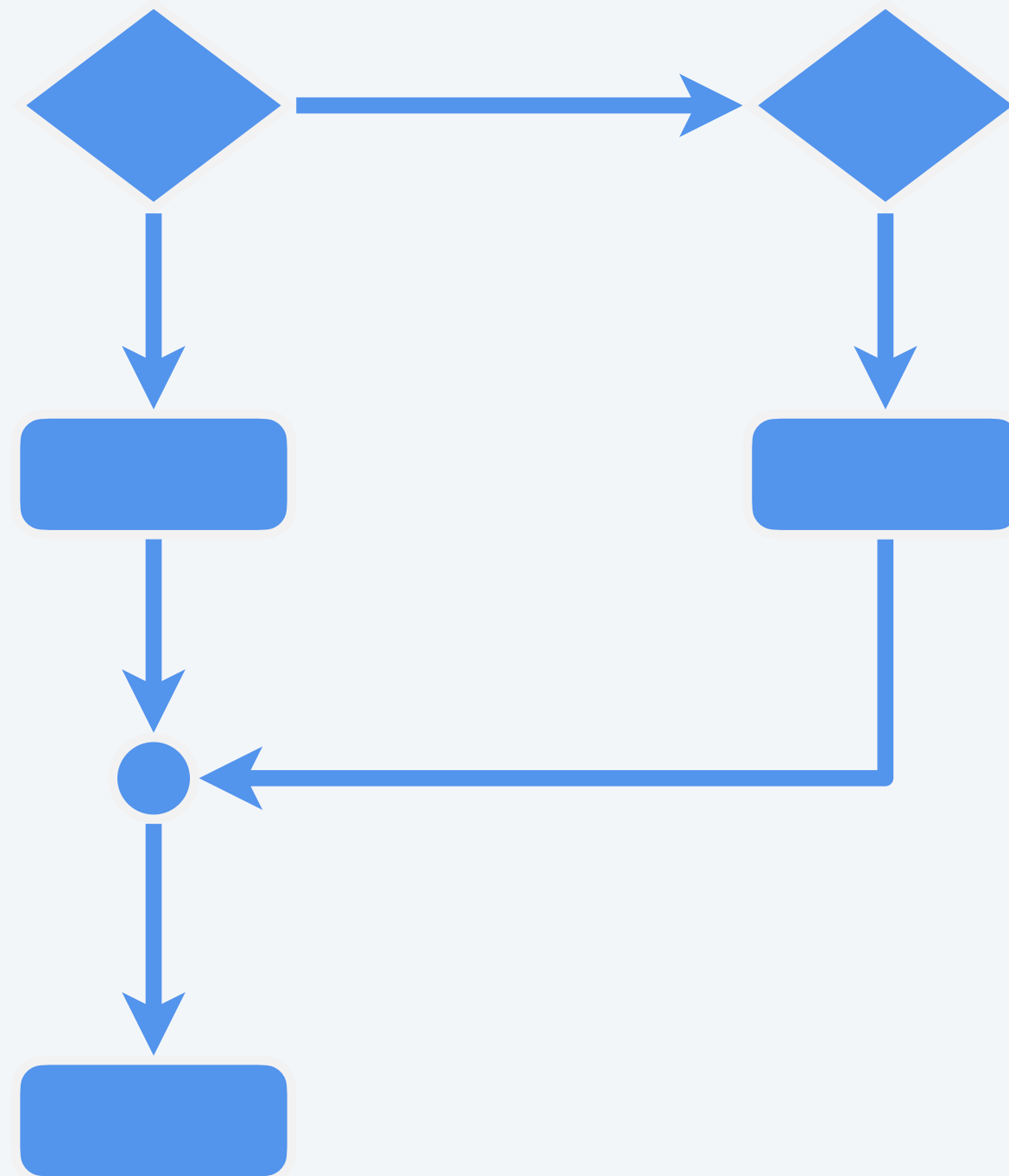
# Summary

---

One-way selection. The *if* statement.

Binary selection. The *if-else* statement.

Multiway selection. Ladder of nested *if-else* statements.



control flow with conditionals

# Credits

---

<b>media</b>	<b>source</b>	<b>license</b>
<i>Decision Making</i>	<a href="https://nextlevelscoaching.com">nextlevelscoaching.com</a>	<a href="#">non-commercial use</a>
<i>Scientific Calculator</i>	<a href="#">Fornax at Wikimedia</a>	<a href="#">CC BY-SA 3.0</a>
<i>Coin Toss</i>	<a href="https://clipground.com">clipground.com</a>	<a href="#">CC BY 4.0</a>
<i>Types of Triangles</i>	<a href="#">Adobe Stock</a>	<a href="#">education license</a>
<i>Bugs</i>	<a href="#">Adobe Stock</a>	<a href="#">education license</a>
<i>Russian Nesting Dolls</i>	<a href="#">Adobe Stock</a>	<a href="#">education license</a>
<i>Rock, Paper, Scissors</i>	<a href="#">Adobe Stock</a>	<a href="#">education license</a>
<i>Watering Can</i>	<a href="#">Katerina Kamprani</a>	
<i>Digital Clock</i>	<a href="#">Chrkl at Wikimedia</a>	<a href="#">CC BY 3.0</a>
<i>Live Coding Icon</i>	<a href="#">Adobe Stock</a>	<a href="#">education license</a>
<i>Code Testing Icon</i>	<a href="#">Adobe Stock</a>	<a href="#">education license</a>
<i>The Year in English</i>	<a href="#">Woodward English</a>	