

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

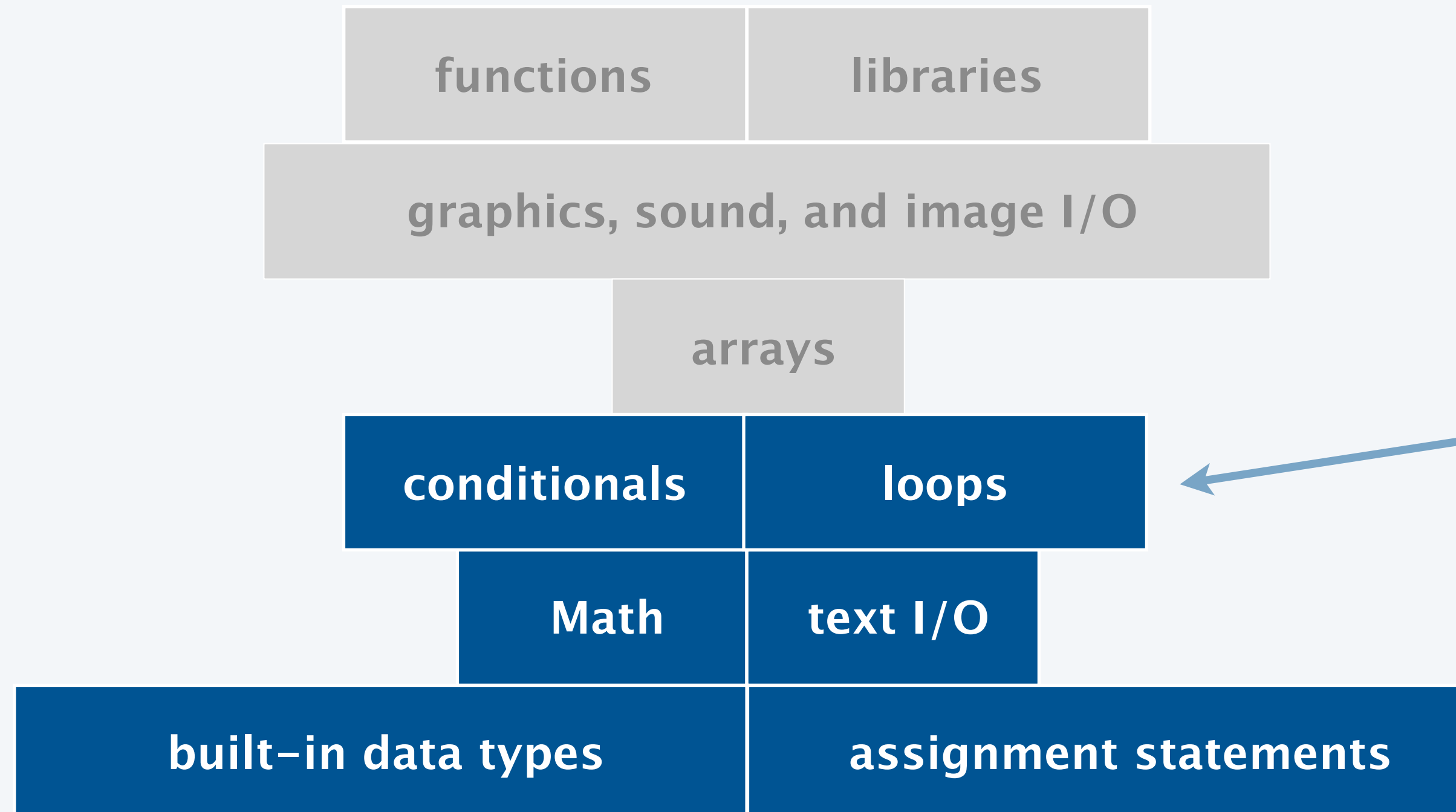
## 1.3 LOOPS

---

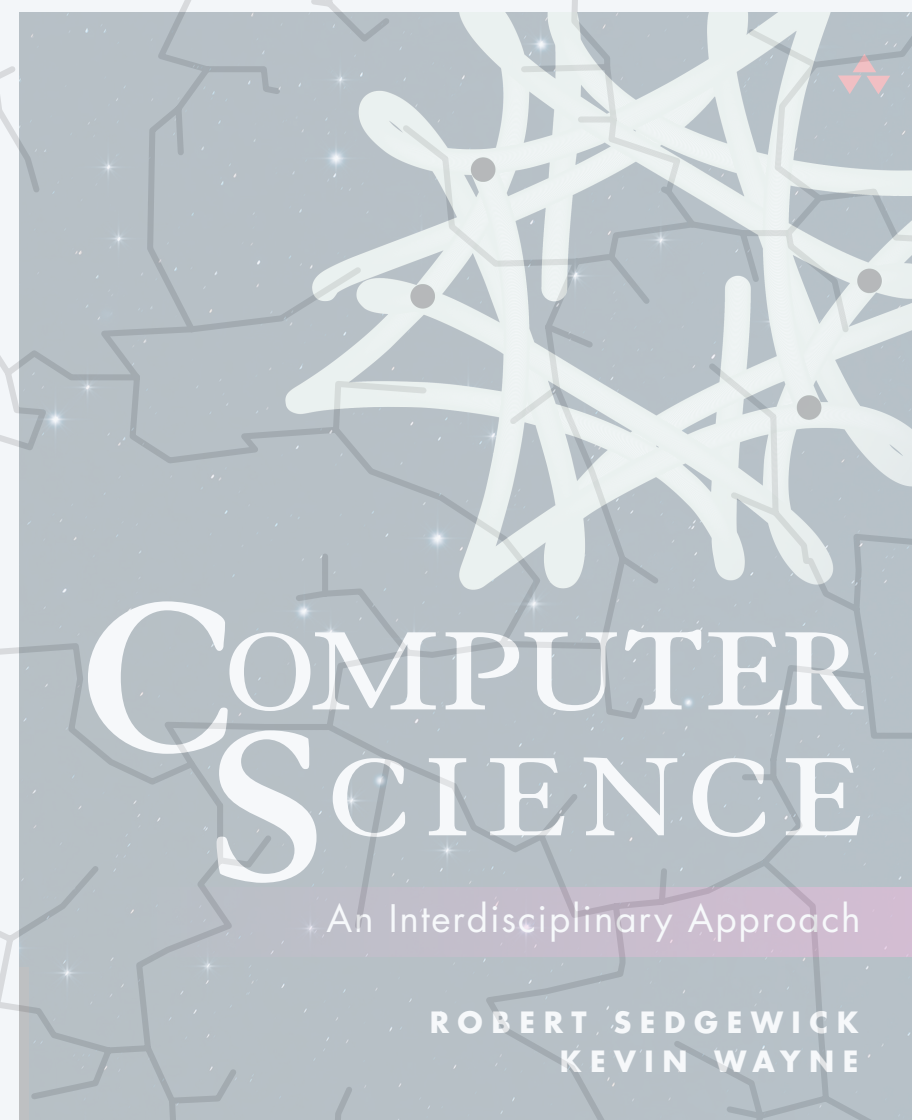
- ▶ *while loops*
- ▶ *do-while loops*
- ▶ *for loops*
- ▶ *nested loops*
- ▶ *image processing*

# Basic building blocks for programming

---



*to infinity and beyond !*



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

## 1.3 LOOPS

---

- ▶ *while loops*
- ▶ *do-while loops*
- ▶ *for loops*
- ▶ *nested loops*
- ▶ *image processing*

# The *while* loop

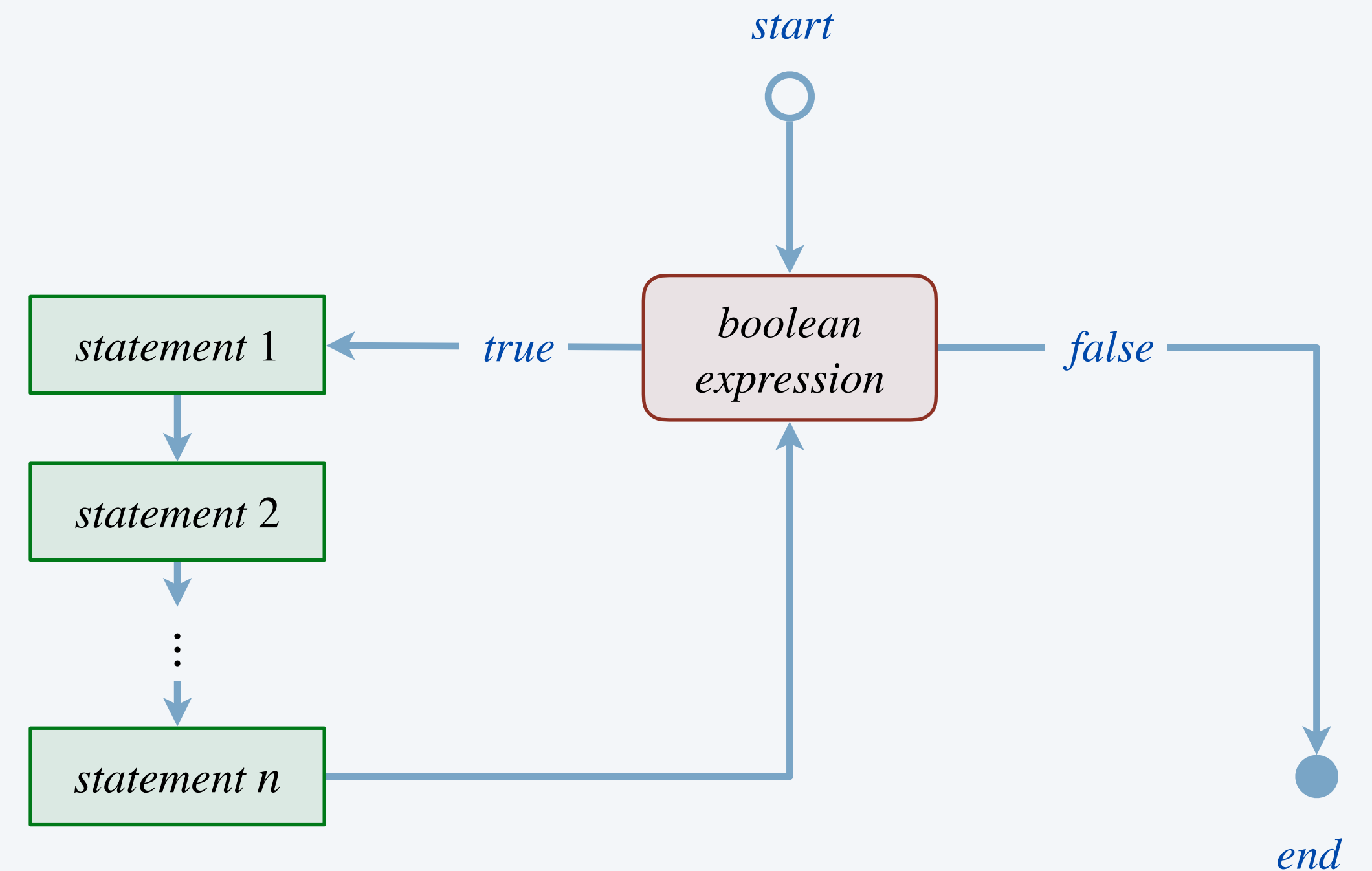
**Goal.** Repeat a certain statement (or statements).

- Evaluate a **boolean expression**. If *true*,
  - execute sequence of statements in **code block**
  - repeat

*loop-continuation condition*

```
while (<boolean expression>) {  
  <statement 1>  
  <statement 2>  
  ⋮  
  <statement n>  
}
```

while loop

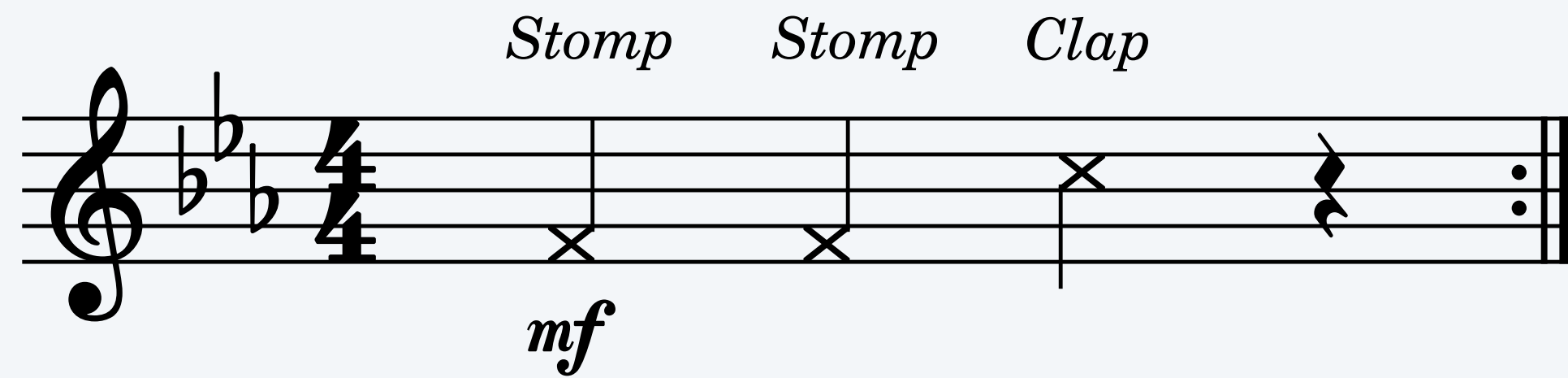


while loop flow chart

# An infinite *while* loop



**Goal.** Recreate percussive beat from Queen's "We Will Rock You."



effect	audio file	sound
<i>stomp</i>	stomp.wav	
<i>clap</i>	clap.wav	
<i>silence</i>	rest.wav	

```
public class StompStompClap {  
    public static void main(String[] args) {  
        while (true) {  
            StdAudio.play("stomp.wav");  
            StdAudio.play("stomp.wav");  
            StdAudio.play("clap.wav");  
            StdAudio.play("rest.wav");  
        }  
    }  
}
```

← an infinite loop



```
~/cos126/loops> java-introcs StompStompClap
```

```
 [plays stomp-stomp-clap beat]
```

← <Ctrl-C> to break out of infinite loop

# Counting from 1 to n



Goal. Repeat a ringtone  $n$  times.



```
public class Ringtone {
    public static void main(String[] args) {
        String filename = args[0];
        int n = Integer.parseInt(args[1]);

        int i = 0;
        while (i < n) {
            StdAudio.play(filename); ← repeat n times
            i++;
        }
    }
}
```

*shorthand for*  
`i = i + 1;`

```
~/cos126/loops> java-introcs Ringtone marimba.wav 1
```

```
🔊 [plays marimba ringtone once]
```

```
~/cos126/loops> java-introcs Ringtone marimba.wav 3
```

```
🔊 [plays marimba ringtone three times]
```

```
~/cos126/loops> java-introcs Ringtone sonar.wav 2
```

```
🔊 [plays sonar ringtone twice]
```

# Counting from 1 to n





**Goal.** Repeat a ringtone  $n$  times.

**Trace.** Show values of variables at end of each iteration of *while* loop.

```
public class Ringtone {
    public static void main(String[] args) {
        String filename = args[0];
        int n = Integer.parseInt(args[1]);

        int i = 0;
        while (i < n) {
            StdAudio.play(filename);
            i++;
        }
    }
}
```

	<i>filename</i>	<i>n</i>	<i>i</i>	
	"marimba.wav"	3	0	← before loop
	"marimba.wav"	3	1	
	"marimba.wav"	3	2	
	"marimba.wav"	3	3	← after loop

**a trace of variables**  
**(values at end of each loop iteration)**



What does the following program do when  $n = 10$ ?

- A. Prints 0 to 10.
- B. Prints even numbers, from 0 to 10.
- C. Prints squares, from 0 to 10.
- D. Prints powers of 2, from  $2^0$  to  $2^{10}$ .
- E. None of the above.

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



# Examples of *while* loops

computation	while loop
<i>print integers from n down to 1</i>	<pre>int i = n; while (i &gt;= 1) {     System.out.println(i);     i--; ← shorthand for            i = i - 1 }</pre>
<i>infinite loop</i>	<pre>while (true) {     StdAudio.play("heartbeat.wav"); }</pre>
<i>number of decimal digits in positive integer x</i>	<pre>int digits = 0; while (x &gt; 0) {     x = x / 10; ← integer division     digits++; }</pre>

*curly braces are optional here  
since only one statement in body of loop  
(but better style to use curly braces)*

# The Collatz conjecture

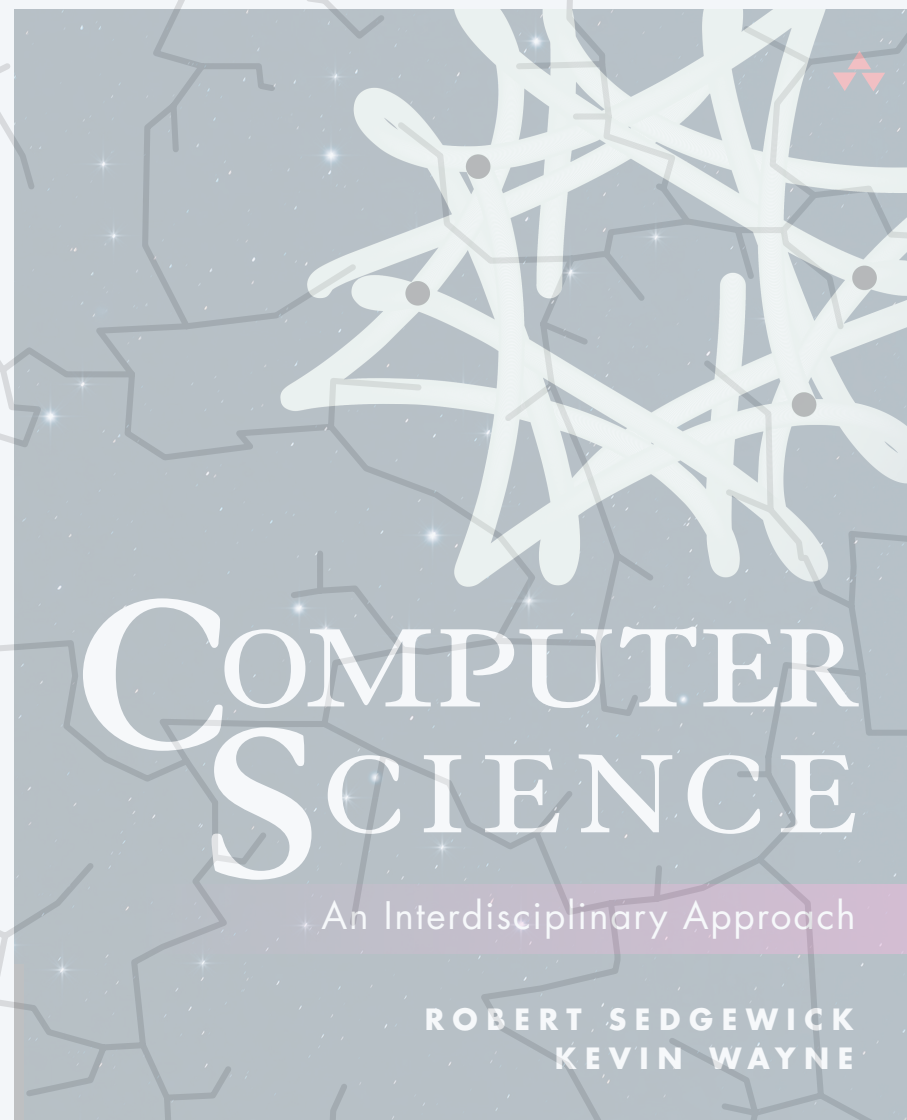


**Goal.** Check if repeated applications of the Collatz transformation yield the number 1.

```
public class Collatz {
    public static void main(String[] args) {
        long n = Long.parseLong(args[0]);

        System.out.println(n);
        while (n != 1) {
            if (n % 2 == 0)
                n /= 2;
            else
                n = n * 3 + 1;
            System.out.println(n);
        }
    }
}
```

Tested up to  $2^{68}$  — larger than `Long.MAX_VALUE`! (But still don't know if always true.)



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

## LOOPS

---

- ▶ *while loops*
- ▶ *do-while loops*
- ▶ *for loops*
- ▶ *nested loops*
- ▶ *image processing*

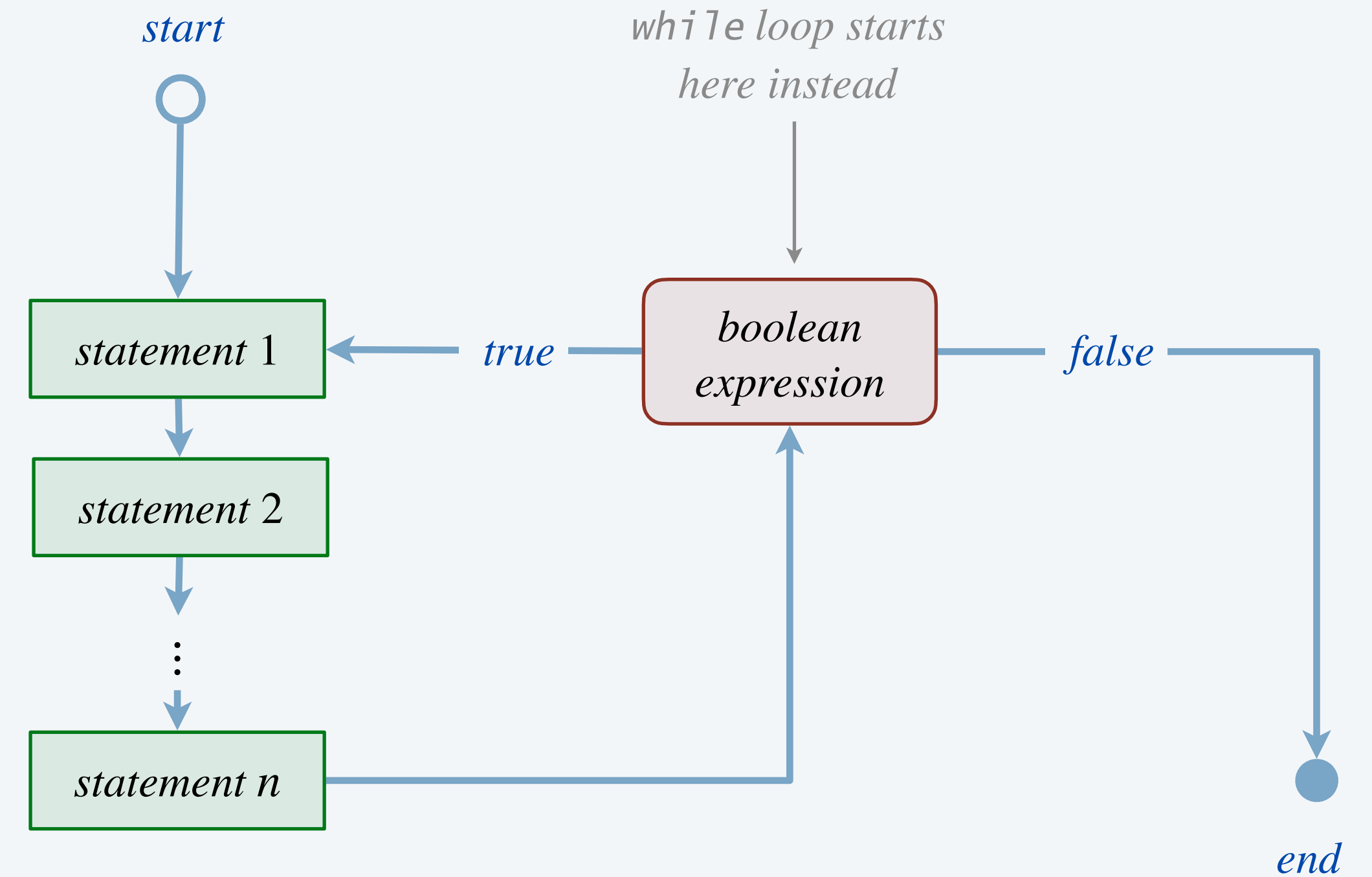
# The *do-while* loop

Another repetition structure.

- Execute a sequence of statements.
- Repeat until some *boolean expression* is true.

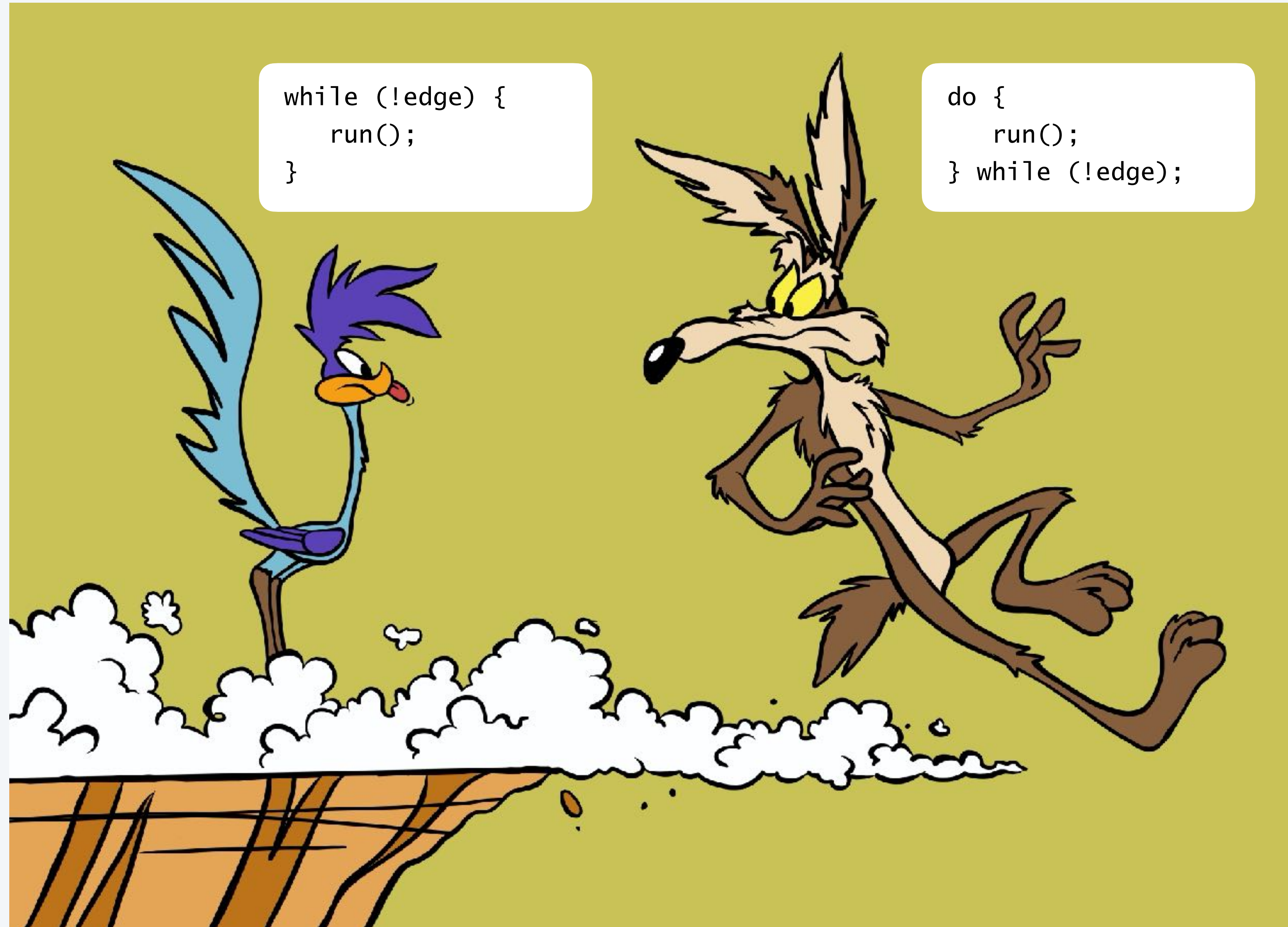
```
do {  
  <statement 1>  
  <statement 2>  
  ⋮  
  <statement n>  
} while (<boolean expression>);
```

do-while loop



do-while loop flow chart

# Wile E. Coyote and Road Runner

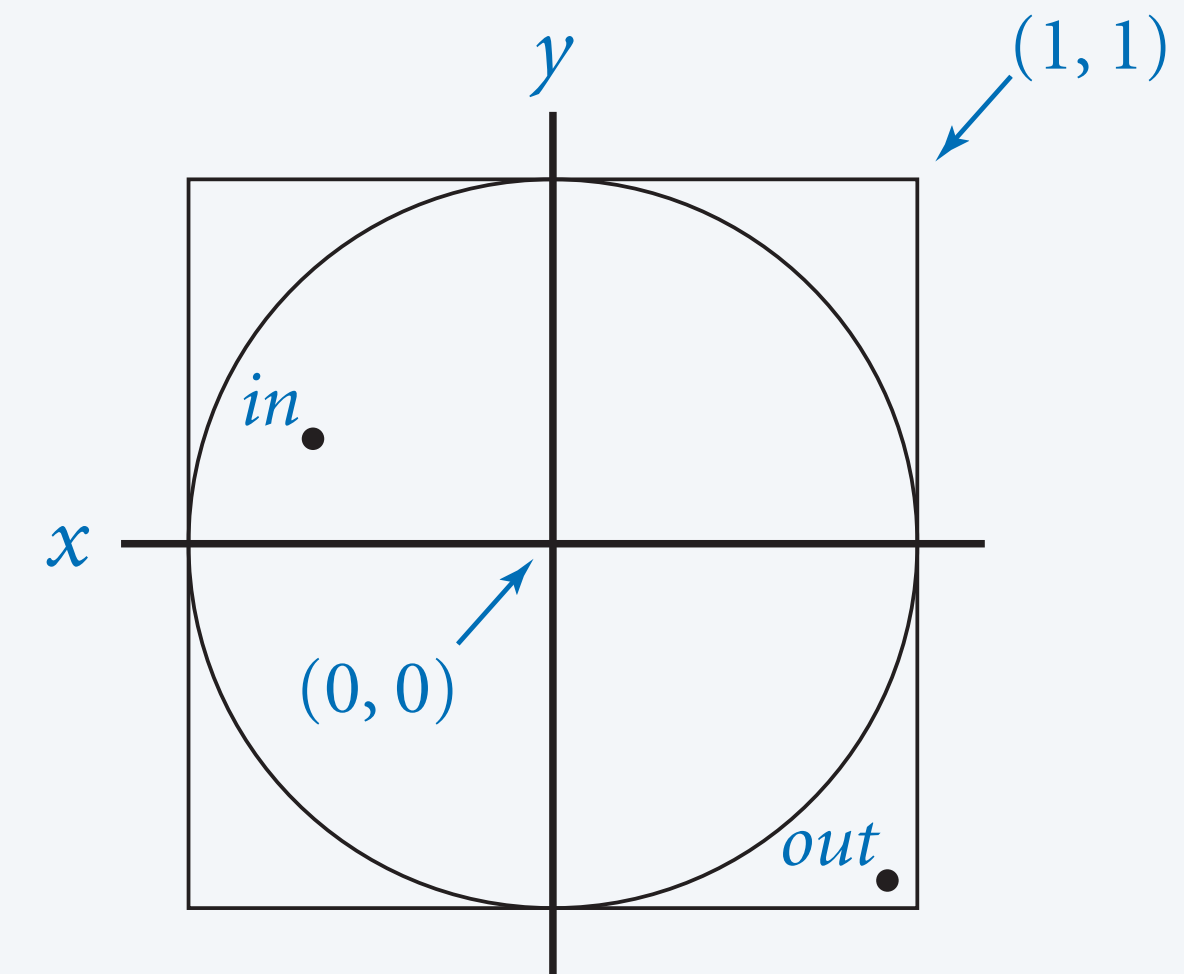


# Random point in unit circle

**Goal.** Generate a random point in unit circle.

**Rejection sampling.**

- Generate a random point in 2-by-2 square centered at origin.
- If point is inside circle, use that point; otherwise, repeat.



```
double x, y; ← must be declared outside block
```

```
do {
```

```
    x = 2.0 * Math.random() - 1.0;
```

```
    y = 2.0 * Math.random() - 1.0;
```

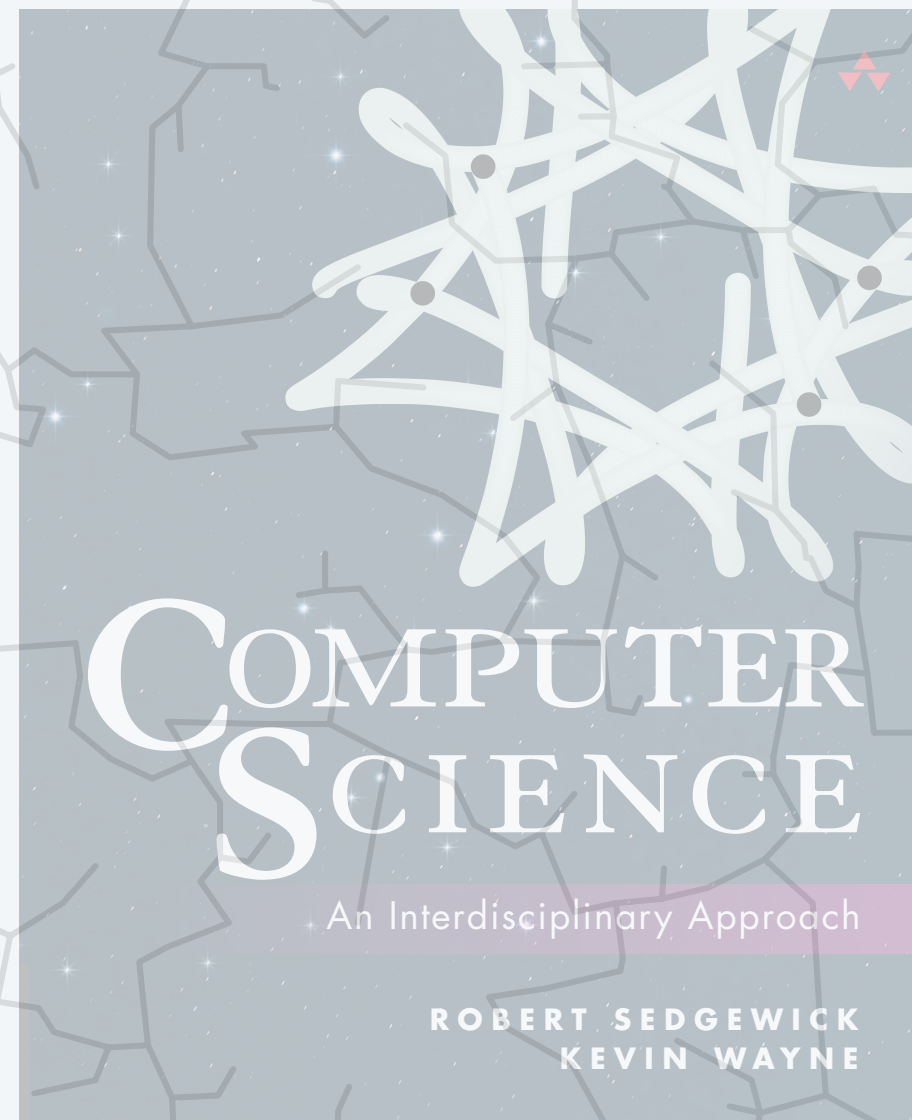
```
} while (x*x + y*y > 1.0);
```

```
System.out.println("(" + x + ", " + y + ")");
```

← random (x, y) in square

← repeat until it's in the circle

**do-while loop**



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

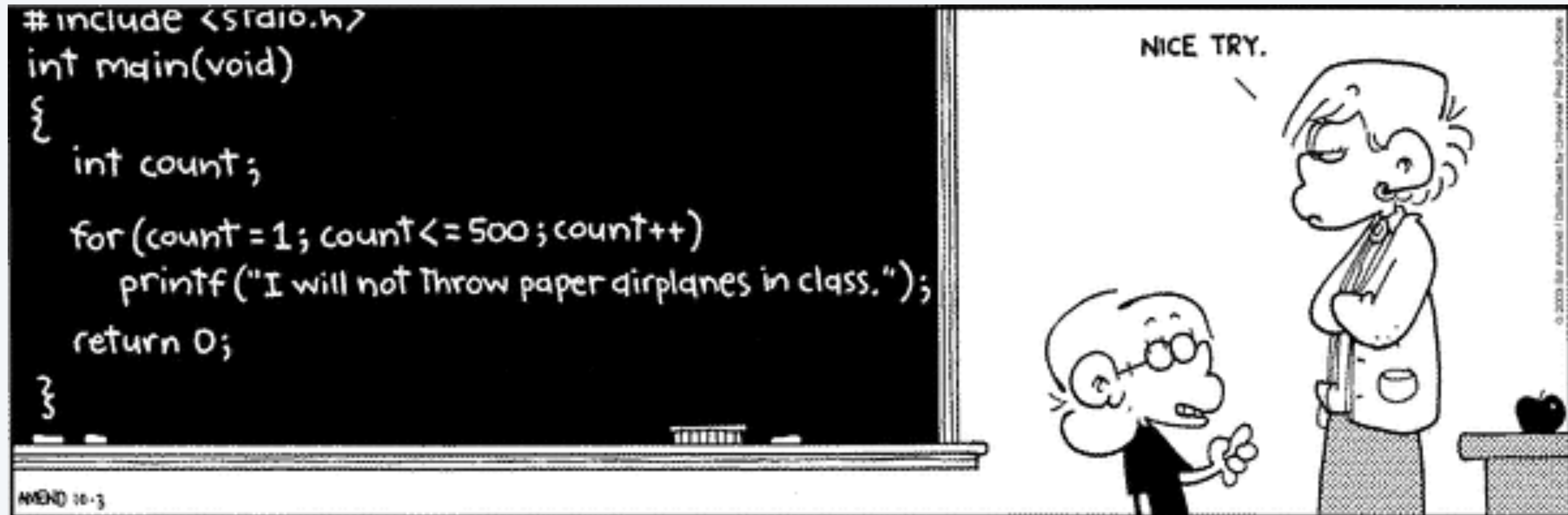
## LOOPS

---

- ▶ *while loops*
- ▶ *do-while loops*
- ▶ ***for loops***
- ▶ *nested loops*
- ▶ *image processing*

# A for loop (in C)

---



Copyright 2004, FoxTrot by Bill Amend



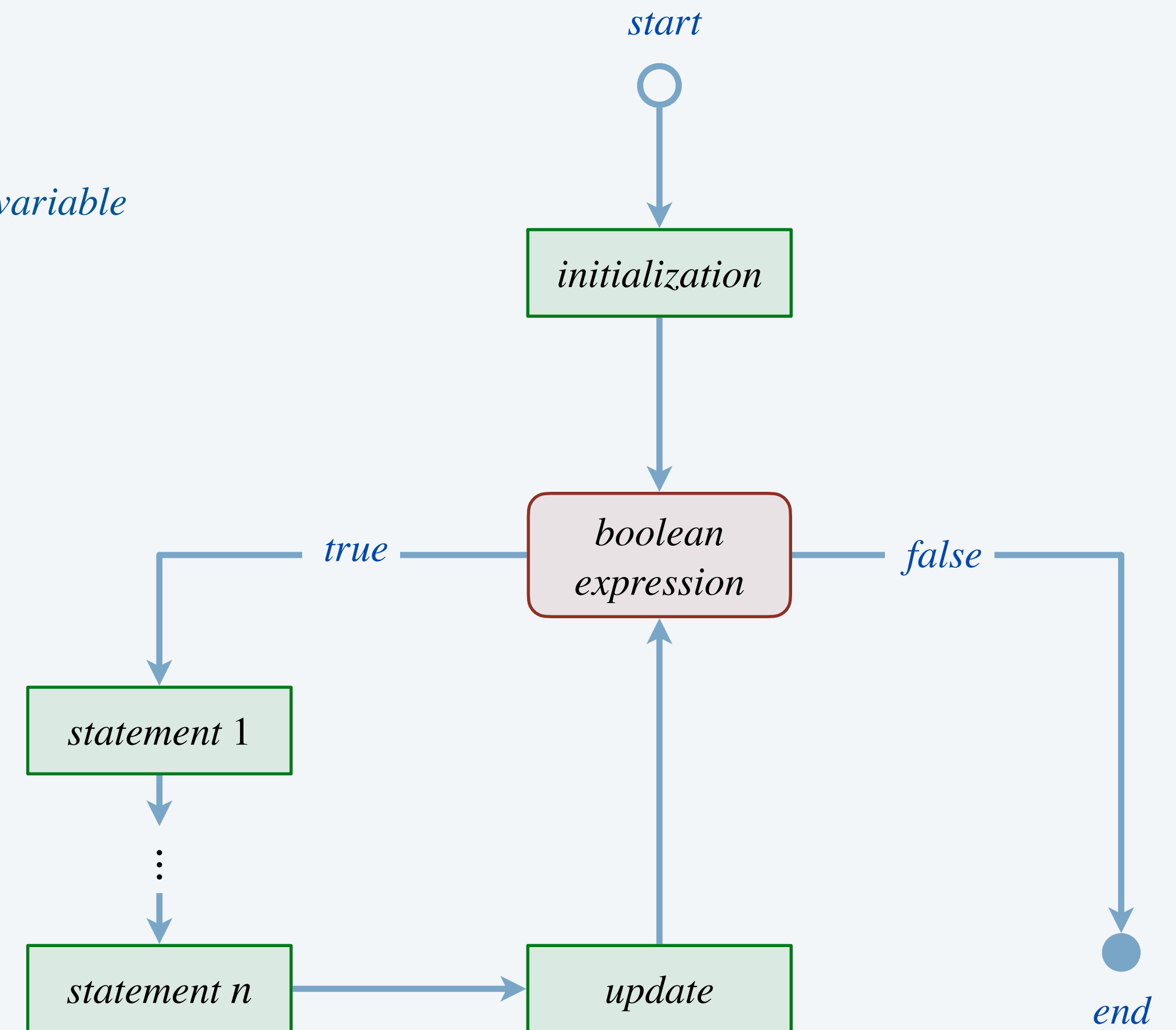
# The *for* loop

An alternative repetition structure.

- Perform an **initialization** step. ← typically, declaring and initializing the value of a variable
- Evaluate a **boolean expression**. If *true*,
  - execute sequence of statements in code block
  - perform an **update** step ← typically, updating the value of a variable
  - repeat

```
for (<init>; <boolean expression>; <update>) {  
  <statement 1>  
  <statement 2>  
  ⋮  
  <statement n>  
}
```

for loop template



for loop flowchart

# Counting from 1 to n



Goal. Play a WAV file  $n$  times. ← *identical behavior as Ringtone.java*

```
public class MusicLoop {
    public static void main(String[] args) {
        String filename = args[0];
        int n = Integer.parseInt(args[1]);

        for (int i = 0; i < n; i++) {
            StdAudio.play(filename);
        }
    }
}
```

*repeat n times*

```
~/cos126/loops> java-introcs MusicLoop heartbeat.wav 1
🔊 [plays heartbeat once]

~/cos126/loops> java-introcs MusicLoop heartbeat.wav 9999999
🔊 [plays heartbeat repeatedly]

~/cos126/loops> java-introcs MusicLoop AmenBreak.wav 10
🔊 [plays The Winstons "Amen Break" drum break 10 times]
```

*among most sampled tracks  
in music history*

# Examples of *for* loops

computation	for loop
<i>factorial</i> $(1 \times 2 \times 3 \times \dots \times n)$	<pre>int product = 1; for (int i = 2; i &lt;= n; i++) {     product *= i; }</pre>
<i>print integers</i> <i>from n down to 1</i>	<pre>for (int i = n; i &gt; 0; i--) {     System.out.println(i); }</pre>
<i>infinite loop</i>	<pre>for (;;) {     StdAudio.play("heartbeat.wav"); }</pre>

← curly braces are optional since  
if body consists of only one statement  
(but better style to include)

← empty initialization and update  
(but better style to use while loop)



**Q.** Which value does the following program print when  $n = 3$ ?

- A.** 8
- B.** 64
- C.** 256
- D.** 512
- E.** 1024

```
public class AnotherMystery {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);

        long result = 2;
        for (int i = 0; i < n; i++)
            result = result * result;

        System.out.println(result);
    }
}
```

## *while* vs. *for* loops

---

**Fact.** Any *while* loop can be replaced with a *for* loop, and vice versa.

**Q.** Which one should I use?

**A.** Guiding principle: use loop construct that leads to clearer code.

**Rule-of-thumb.** Use a *for* loop when you know the number of iterations ahead of time.

```
int i = 0;
while (i < n) {
    StdAudio.play(filename);
    i++;
}
```

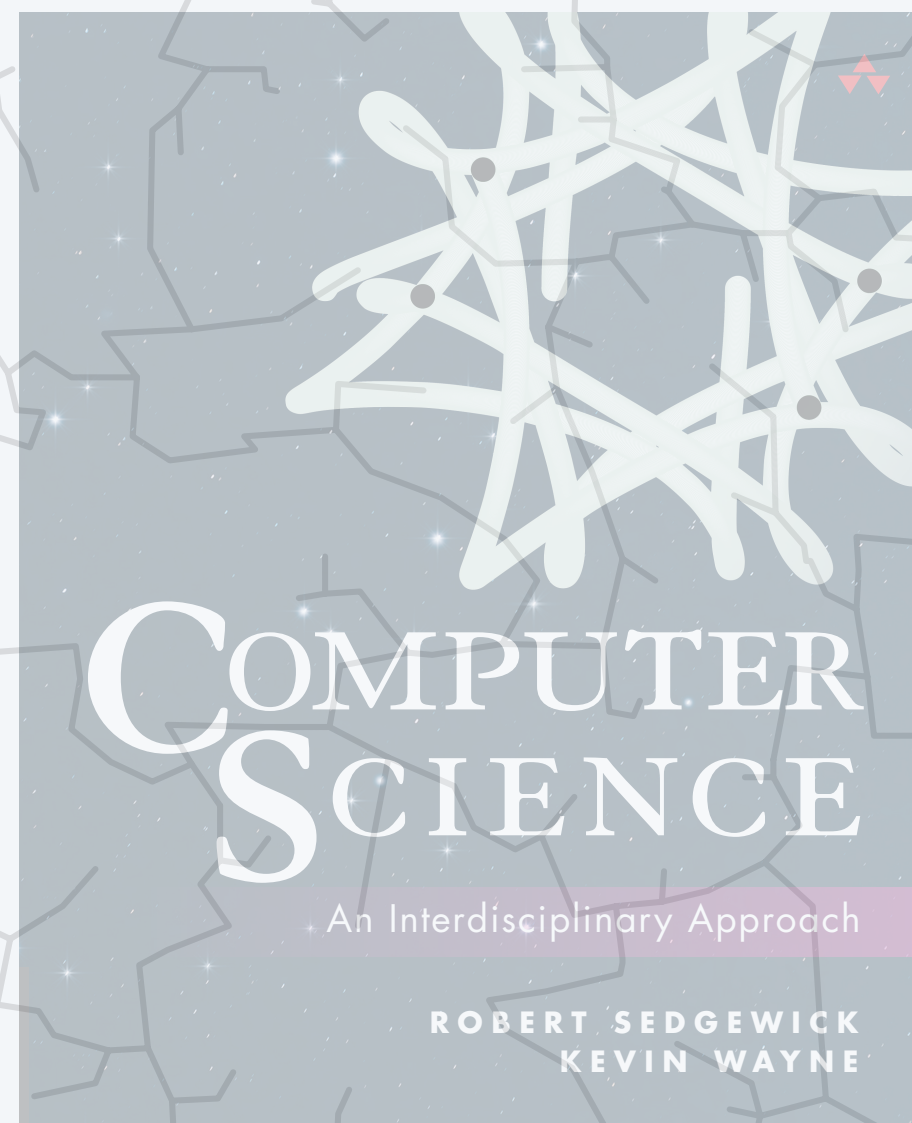
while loop

```
for (int i = 0; i < n; i++) {
    StdAudio.play(filename);
}
```

equivalent for loop  
(except *i* not accessible after loop)

*code controlling loop  
localized to one place*





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

## LOOPS

---

- ▶ *while loops*
- ▶ *do-while loops*
- ▶ *for loops*
- ▶ *nested loops*
- ▶ *image processing*



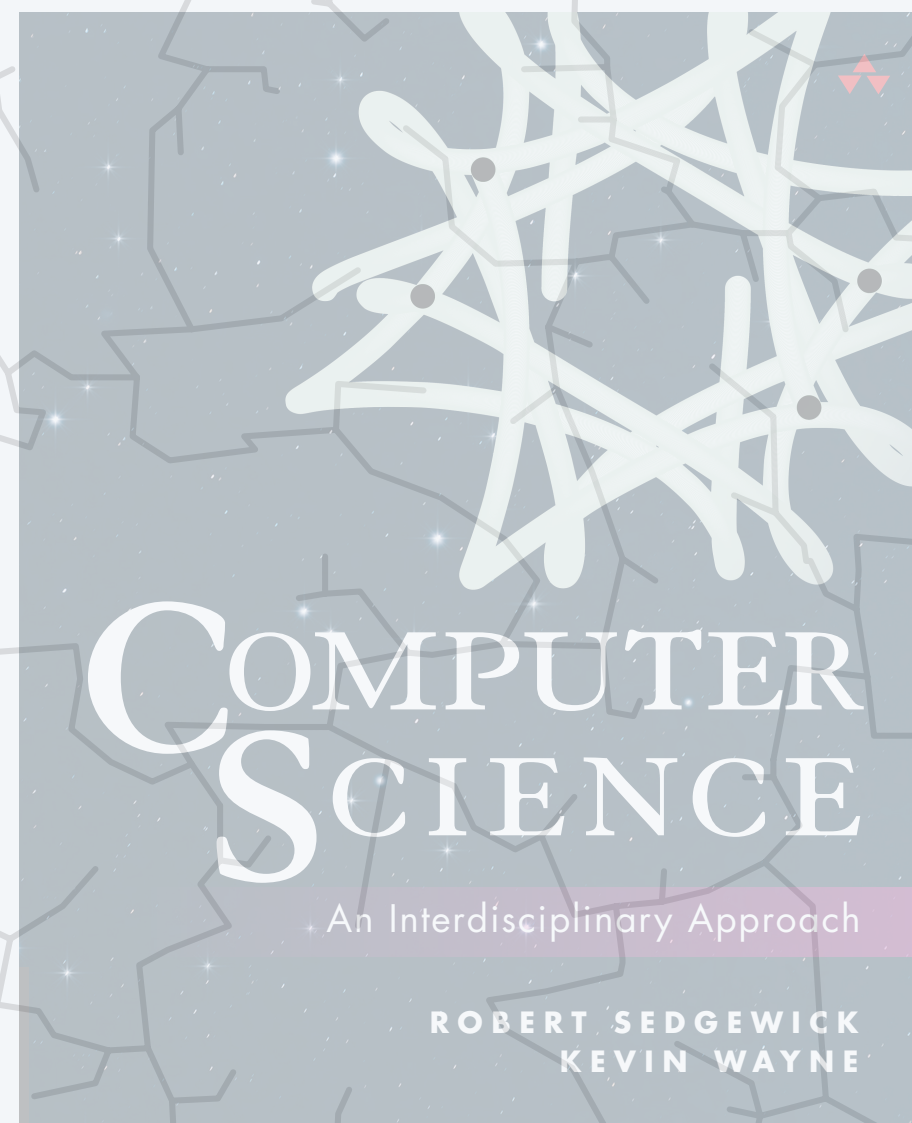


Suppose  $m = 4$  and  $n = 7$ . How many lines of output does the following program produce?

- A. 4
- B. 7
- C. 13
- D. 28
- E. 32

```
public class YetAnotherMystery {  
    public static void main(String[] args) {  
        int m = Integer.parseInt(args[0]);  
        int n = Integer.parseInt(args[1]);  
  
        for (int i = 0; i < m; i++) {  
            for (int j = 0; j < n; j++) {  
                System.out.println(i + ", " + j);  
            }  
        }  
    }  
}
```

*for loop nested  
within a for loop*

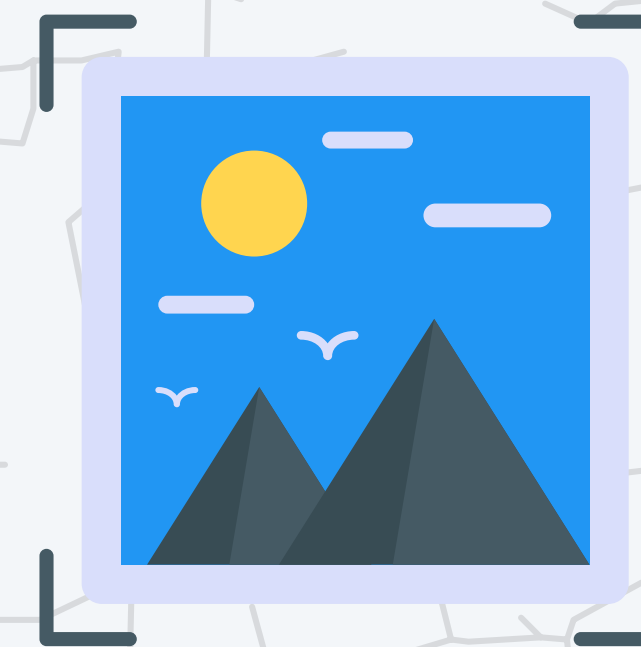


<https://introcs.cs.princeton.edu>

## LOOPS

---

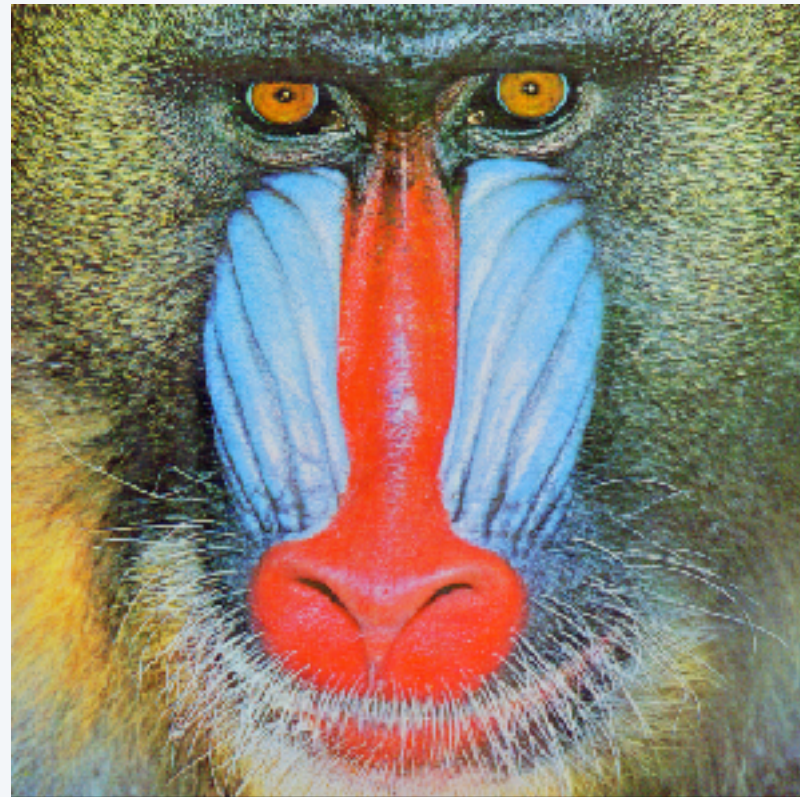
- ▶ *while loops*
- ▶ *do-while loops*
- ▶ *for loops*
- ▶ *nested loops*
- ▶ ***image processing***





# Image processing

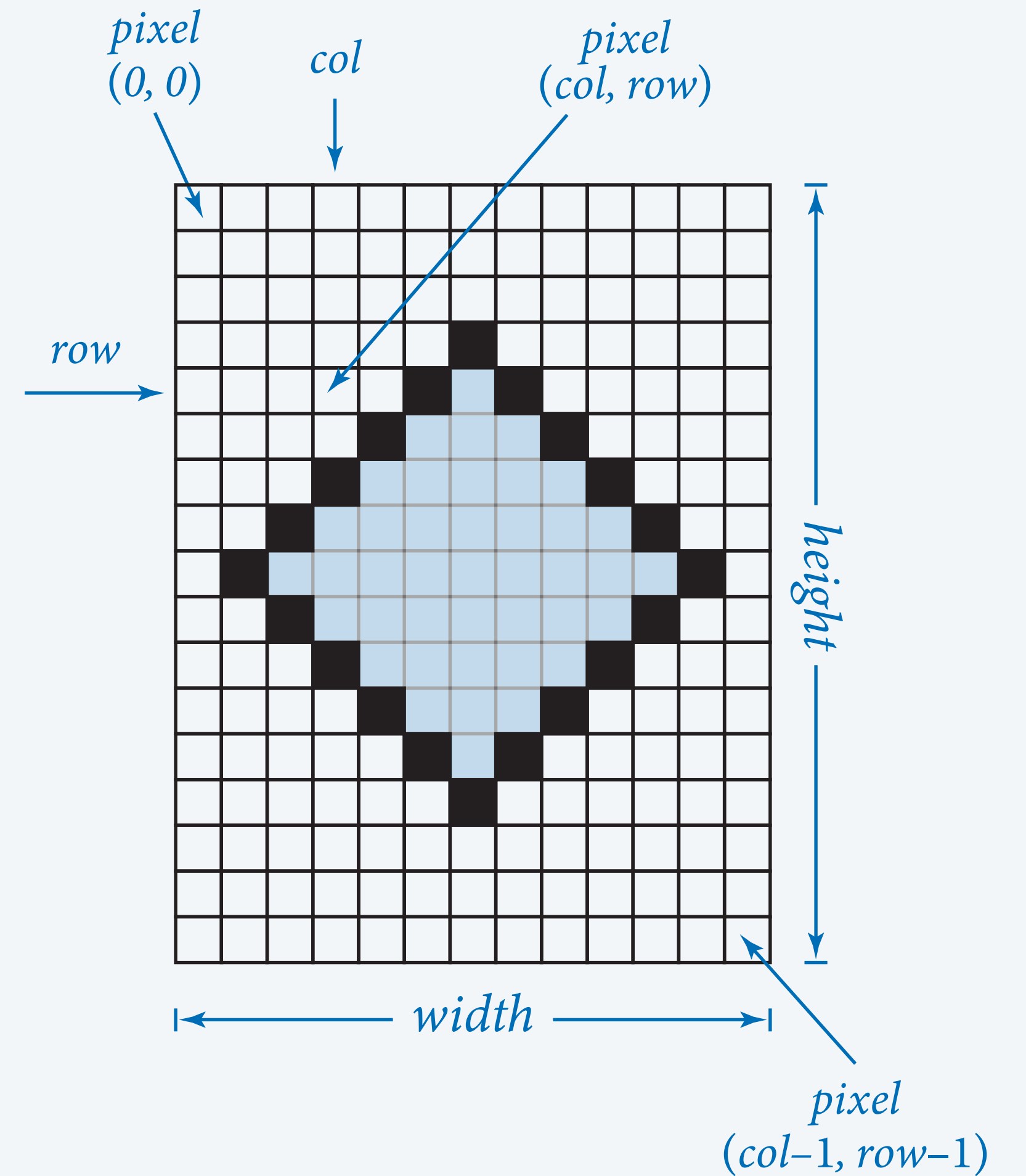
A **picture** is a *width-by-height* grid of pixels; each pixel has a **color**.



mandrill.jpg



arch.jpg



## Image-processing conventions.

- Pixel  $(i, j)$  means column  $i$  and row  $j$ .
- Pixel  $(0, 0)$  is upper-left.

← warning: different conventions from matrices and Cartesian coordinates

# RGB color model


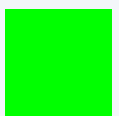


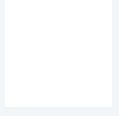
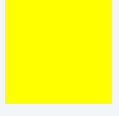

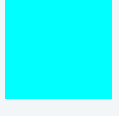
---

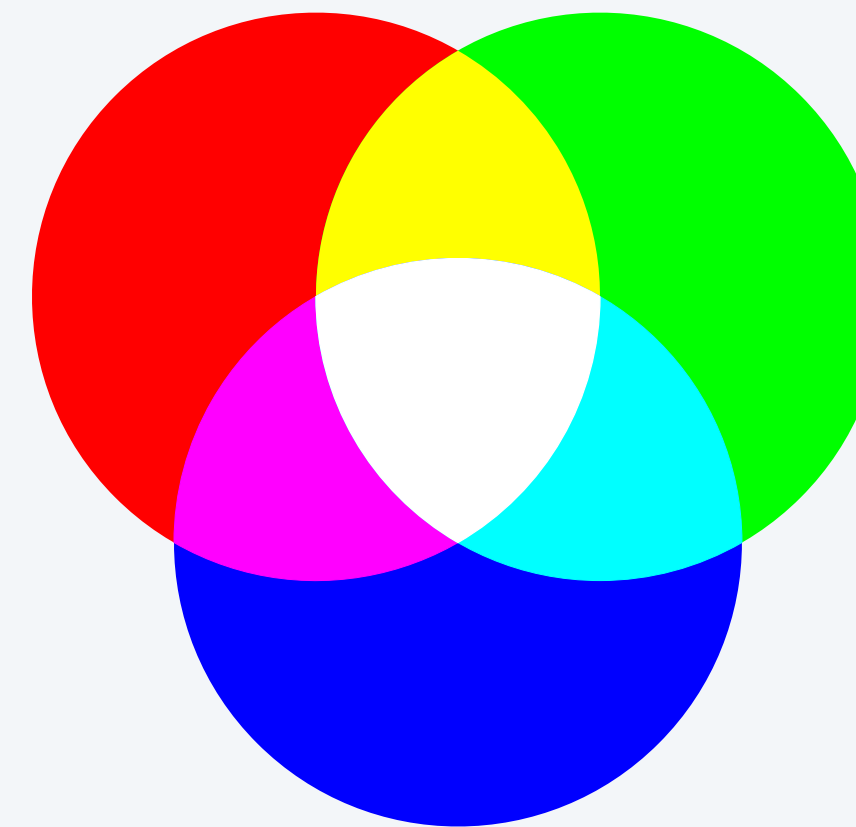
**Color** is a sensation in the eye from electromagnetic radiation.

**RGB color model.** Popular format for representing color on digital displays.

- Color is composed of red, green, and blue components.
- Each color component is an integer between 0 to 255.



<b>name</b>	<b>red</b>	<b>green</b>	<b>blue</b>	<b>color</b>
<i>red</i>	255	0	0	
<i>green</i>	0	255	0	
<i>blue</i>	0	0	255	
<i>black</i>	0	0	0	
<i>white</i>	255	255	255	
<i>yellow</i>	255	255	0	
<i>magenta</i>	255	0	255	
<i>cyan</i>	0	255	255	
<i>book blue</i>	0	64	128	



# Grayscale



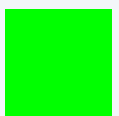
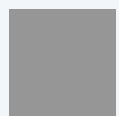




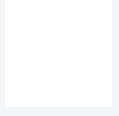
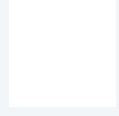
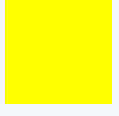
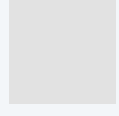

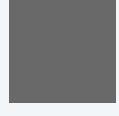
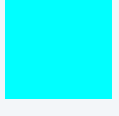
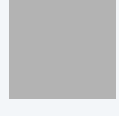

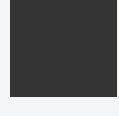
**Goal.** Convert color image to grayscale.

*fundamental operation  
in computer graphics and vision*

- RGB color is a shade of gray when  $R = G = B$ .
- To convert RGB color to grayscale, use **luminance** for  $R$ ,  $G$ , and  $B$  values:

*not the same as in ColorContrast!*  $\longrightarrow Y = 0.299 R + 0.587 G + 0.114 B$



name	red	green	blue	color	lum	gray
<i>red</i>	255	0	0		76	
<i>green</i>	0	255	0		150	
<i>blue</i>	0	0	255		29	
<i>black</i>	0	0	0		0	
<i>white</i>	255	255	255		255	
<i>yellow</i>	255	255	0		226	
<i>magenta</i>	255	0	255		105	
<i>cyan</i>	0	255	255		179	
<i>book blue</i>	0	64	128		52	

$$\begin{aligned}
 Y &= 0.299 R + 0.587 G + 0.114 B \\
 &= 0.299 (0) + 0.587 (64) + 0.114 (128) \\
 &= 52.16
 \end{aligned}$$

# Standard picture library



*StdPicture*. Our library for manipulating images. ← *available with javac-introcs and java-introcs commands*

```
public class StdPicture
```

```
static void read(String filename)
```

*initialize picture from filename*

```
static void save(String filename)
```

*save picture to filename*

```
static int width()
```

*width of picture*

```
static int height()
```

*height of picture*

```
static int getRed(int col, int row)
```

*red component of pixel (col, row)*

```
static int getGreen(int col, int row)
```

*green component of pixel (col, row)*

```
static int getBlue(int col, int row)
```

*blue component of pixel (col, row)*

```
static void setRGB(int col, int row,  
                  int r, int g, int b)
```

*set color of pixel (col, row) to (r, g, b)*

```
⋮
```

```
⋮
```

← *supported file formats:  
JPEG, PNG, GIF, TIFF, BMP*

# Image processing: color image filters

---



original



grayscale



sepia



duotone



brighter



darker



RGB layers



negative

# Grayscale filter

```
public class Grayscale {
    public static void main(String[] args) {
        String filename = args[0];
        StdPicture.read(filename);
        int width = StdPicture.width();
        int height = StdPicture.height();

        for (int col = 0; col < width; col++) {
            for (int row = 0; row < height; row++) {
                int r = StdPicture.getRed(col, row);
                int g = StdPicture.getGreen(col, row);
                int b = StdPicture.getBlue(col, row);
                int y = (int) (Math.round(0.299*r + 0.587*g + 0.114*b));
                StdPicture.setRGB(col, row, y, y, y);
            }
        }

        StdPicture.show();
    }
}
```

*read picture from file  
and get dimensions*

*iterate over  
all pixels*

*get RGB values*

*luminance formula  
( $Y = 0.299R + 0.587G + 0.114B$ )*

*display picture in window*

```
~/> java-introcs Grayscale arch.jpg
```



*luminance formula  
( $Y = 0.299R + 0.587G + 0.114B$ )*

# Compute the negative: demo



**Goal.** Find the negative of an image (where light colors become dark and vice-versa).

**Algorithm.** For each pixel with color values  $(r, g, b)$ , replace it with  $(255 - r, 255 - g, 255 - b)$ .

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)

original image

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)

negative image

# Compute the negative: implementation

---

**Goal.** Find the negative of an image (where light colors become dark and vice-versa).

**Algorithm.** For each pixel with color values  $(r, g, b)$ , replace it with  $(255 - r, 255 - g, 255 - b)$ .

```
for (int row = 0; row < height; row++) {  
    for (int col = 0; col < width; col++) {  
        int r = StdPicture.getRed(col, row);  
        int g = StdPicture.getGreen(col, row);  
        int b = StdPicture.getBlue(col, row);  
  
        StdPicture.setRGB(col, row, 255 - r, 255 - g, 255 - b);  
    }  
}  
StdPicture.show();
```

```
~/> java-introcs Negative arch.jpg
```







What image does the following code fragment produce?

- A. Original image.
- B. Negative.
- C. Red channel.
- D. Blue channel.
- E. Green channel.

```
for (int col = 0; col < width; col++) {  
    for (int row = 0; row < height; row++) {  
        int r = StdPicture.getRed(col, row);  
        int g = StdPicture.getGreen(col, row);  
        int b = StdPicture.getBlue(col, row);  
  
        StdPicture.setRGB(col, row, r, 0, 0);  
    }  
}  
StdPicture.show();
```

← *for loops in reverse order*

← *different arguments to setRGB*



**Goal.** Increase the brightness of every pixel.

**Algorithm.** For each pixel with color values  $(r, g, b)$ , replace it with  $\left(\frac{255 + r}{2}, \frac{255 + g}{2}, \frac{255 + b}{2}\right)$ .

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)

original image

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)

brighter image

# Brighten: implementation

---

**Goal.** Increase the brightness of every pixel.

**Algorithm.** For each pixel with color values  $(r, g, b)$ , replace it with  $\left(\frac{255 + r}{2}, \frac{255 + g}{2}, \frac{255 + b}{2}\right)$ .

```
for (int row = 0; row < height; row++) {
    for (int col = 0; col < width; col++) {
        int r = Math.round((StdPicture.getRed(col, row) + 255) / 2.0);
        int g = Math.round((StdPicture.getGreen(col, row) + 255) / 2.0);
        int b = Math.round((StdPicture.getBlue(col, row) + 255) / 2.0);

        StdPicture.setRGB(col, row, r, g, b);
    }
}
StdPicture.show();
```

```
~/> java-introcs Brighten arch.jpg
```



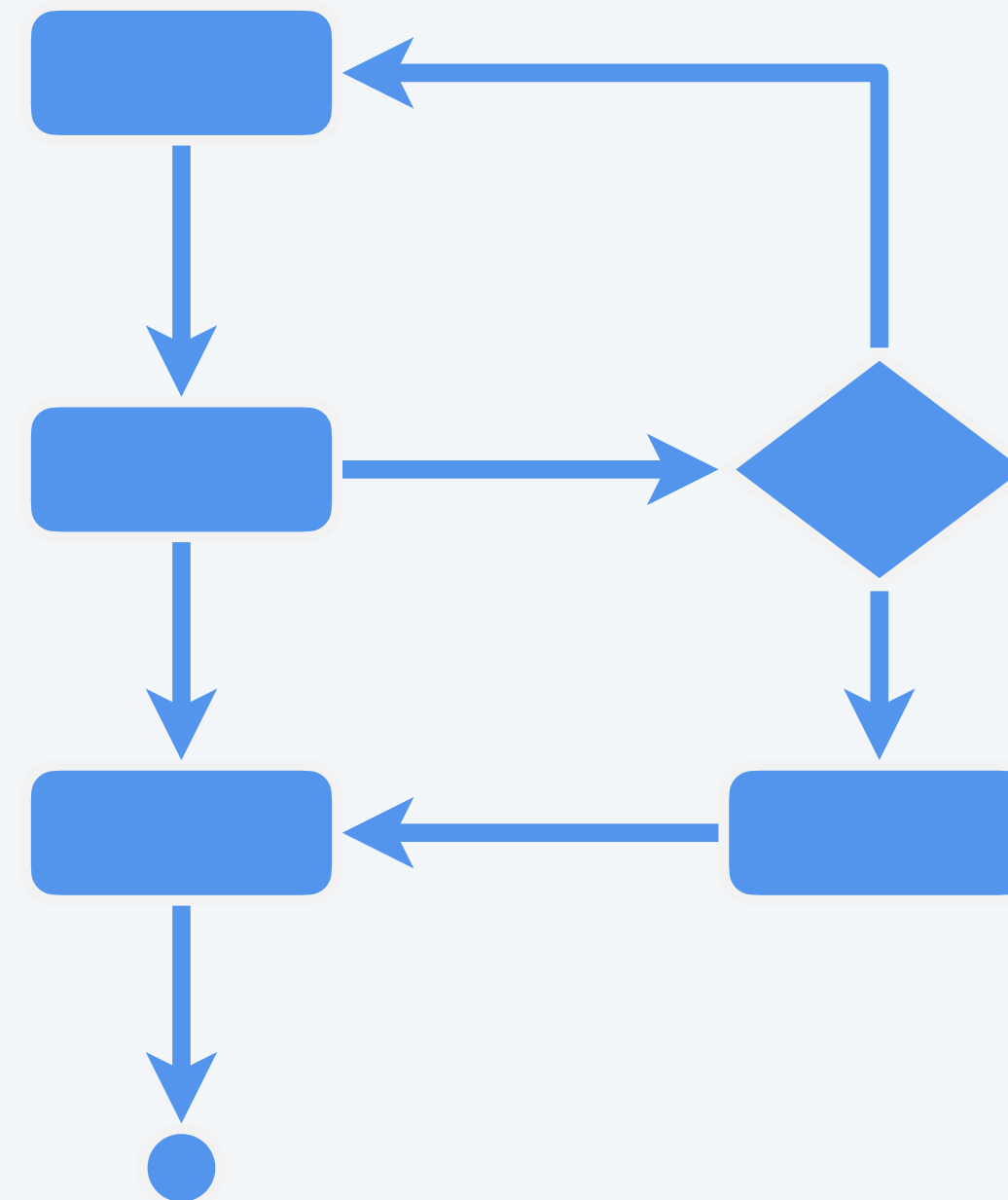
# Summary

---

**Iteration.** Use *while* and *for* loops to repeat code in a program.

**Nested iteration.** Body of loop contains another loop.

**Image processing.** An image is a 2D grid of pixels, each of which has r, g and b color levels.



control flow with conditionals and loops

# Credits

---

<b>media</b>	<b>source</b>	<b>license</b>
<i>Russian Nesting Dolls</i>	<u><a href="#">Adobe Stock</a></u>	<u><a href="#">education license</a></u>
<i>Image Processing Icon</i>	<u><a href="#">Adobe Stock</a></u>	<u><a href="#">education license</a></u>
<i>Mandrill</i>	<u><a href="#">USC SIPI Image Database</a></u>	
<i>Johnson Arch</i>	<u><a href="#">Danielle Alio Capparella</a></u>	by photographer
<i>RGB Color Model</i>	<u><a href="#">Wikimedia</a></u>	<u><a href="#">Kopimi</a></u>
<i>LGBTQ+ Eye</i>	<u><a href="#">Wikimedia</a></u>	<u><a href="#">CC BY 2.0</a></u>