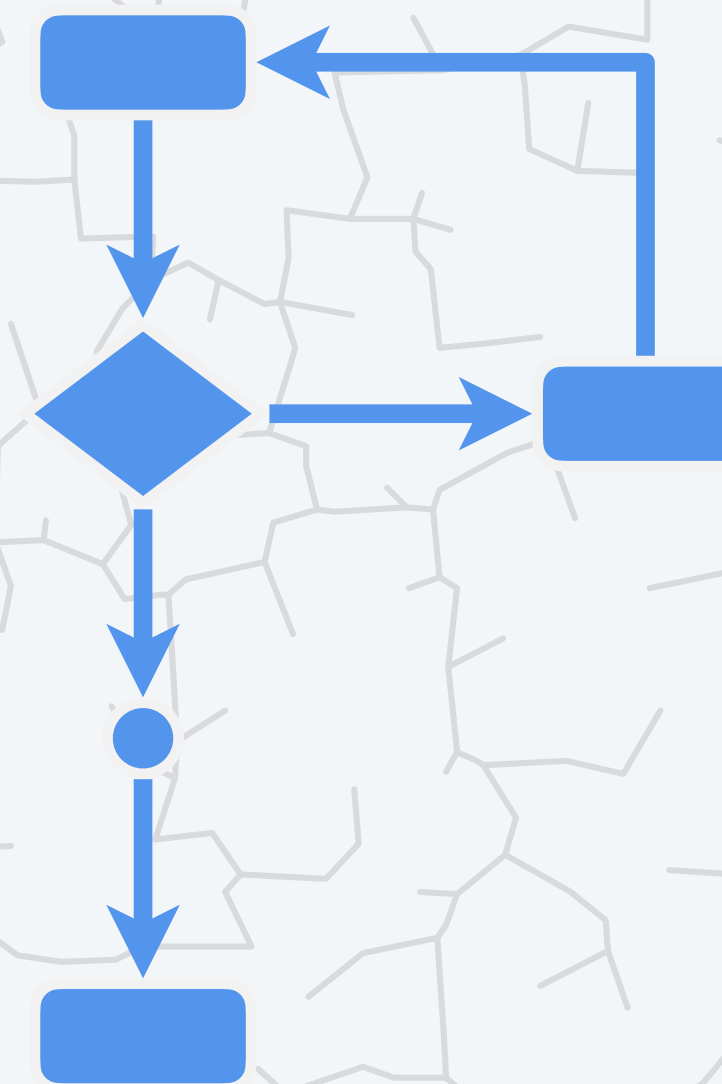


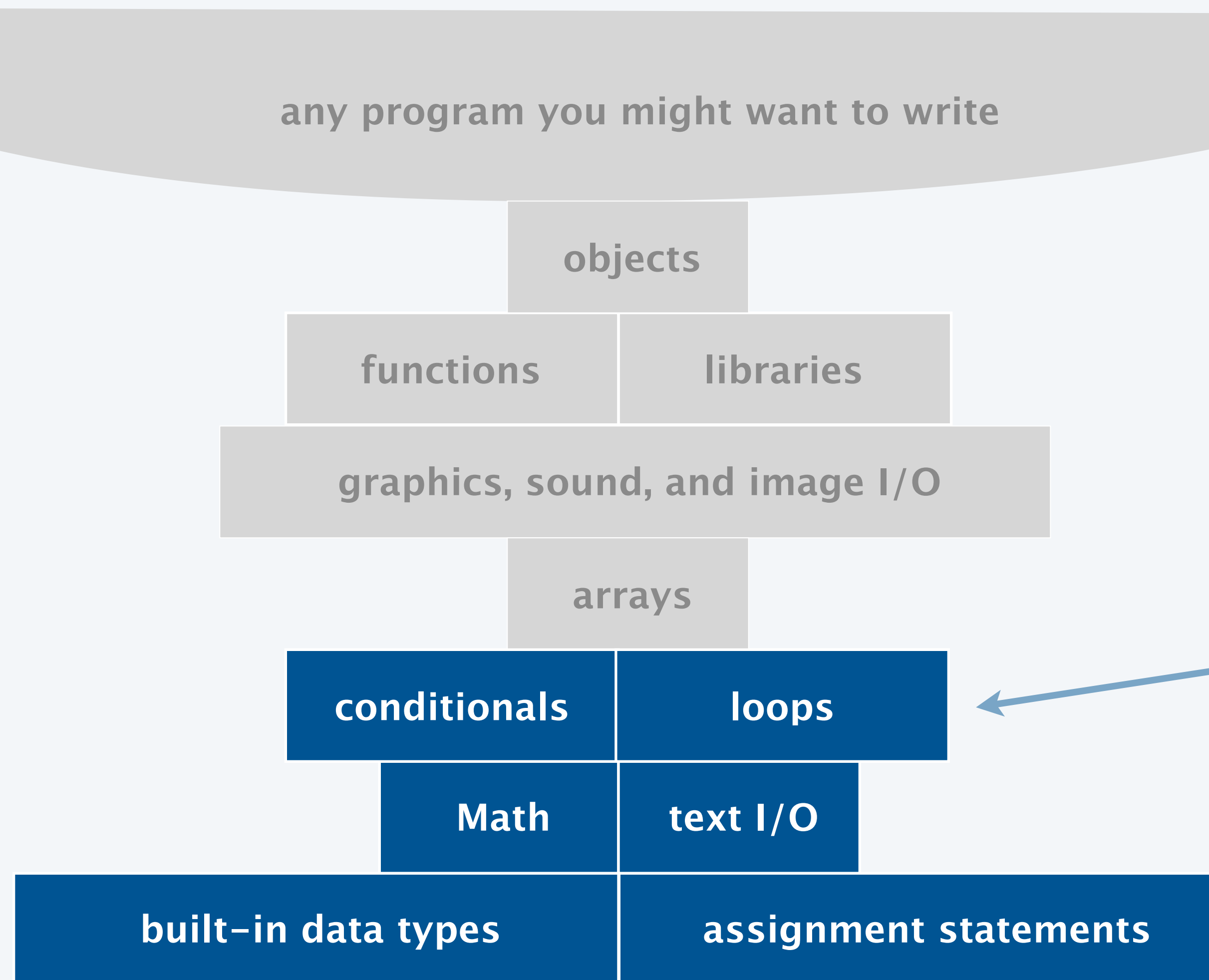
<https://introc.cs.princeton.edu>

1.3 LOOPS

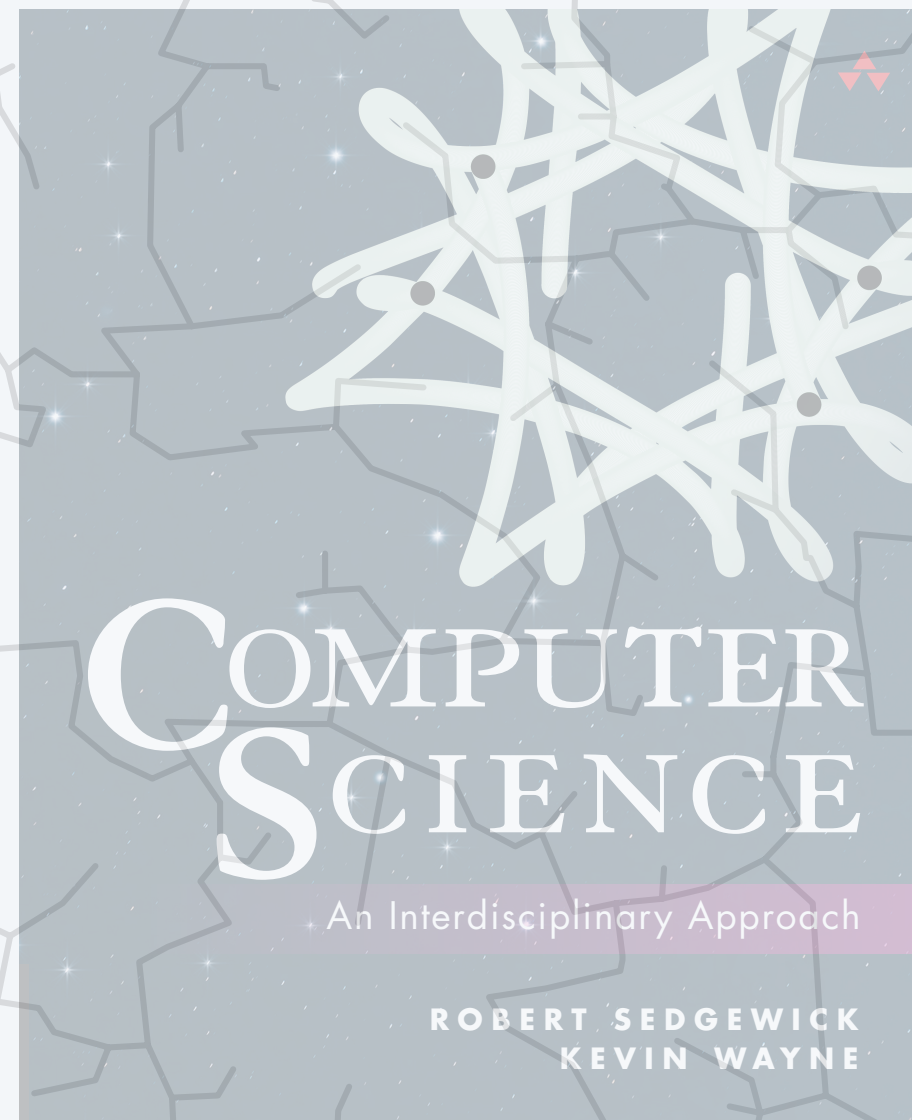
- ▶ *while loops*
- ▶ *for loops*
- ▶ *nested loops*
- ▶ *image processing*



Basic building blocks for programming



to infinity and beyond !



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

1.3 LOOPS

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

The *while* loop

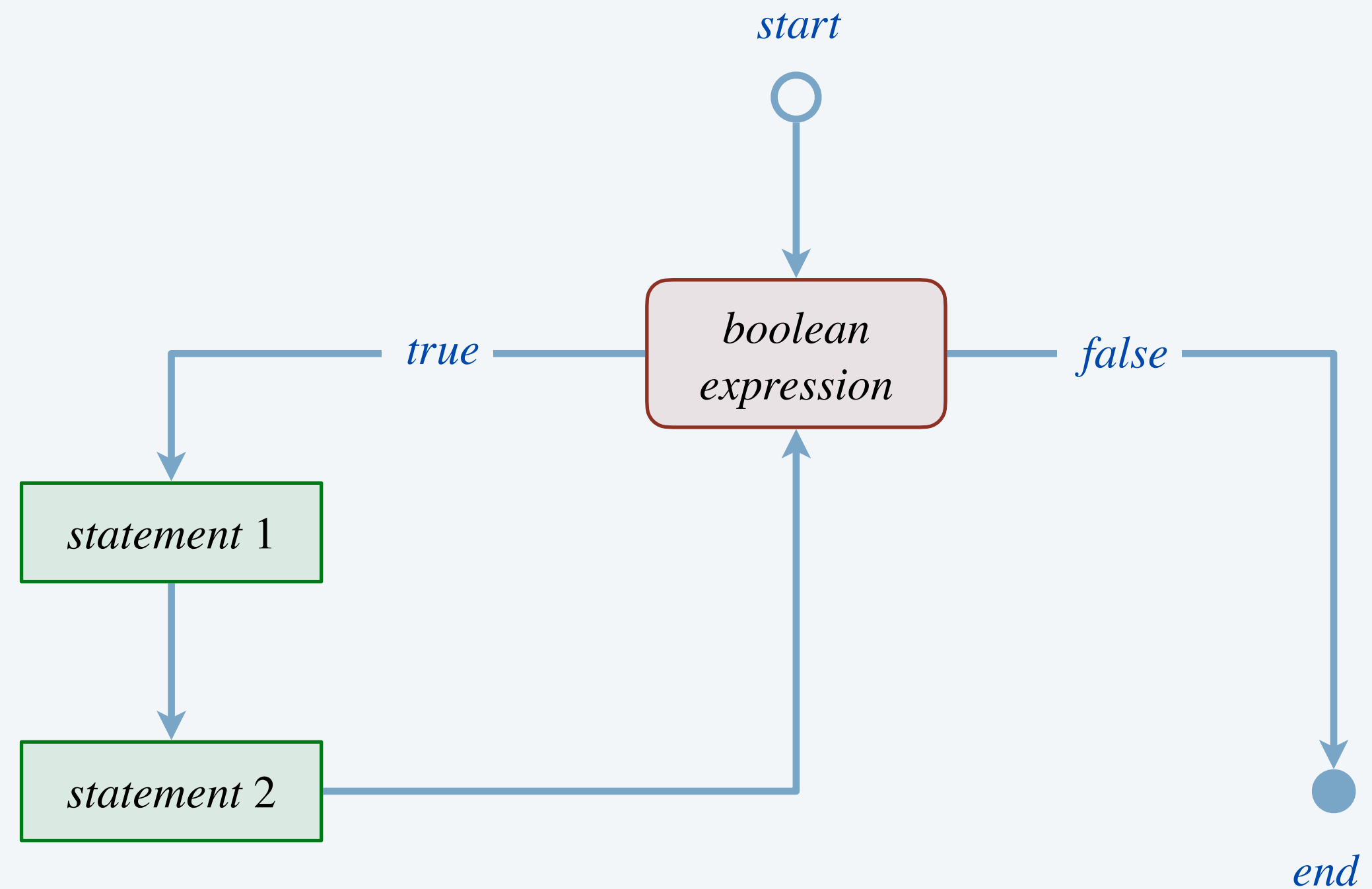
Goal. Repeat a certain statement (or statements).

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

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

loop-continuation condition

while loop template

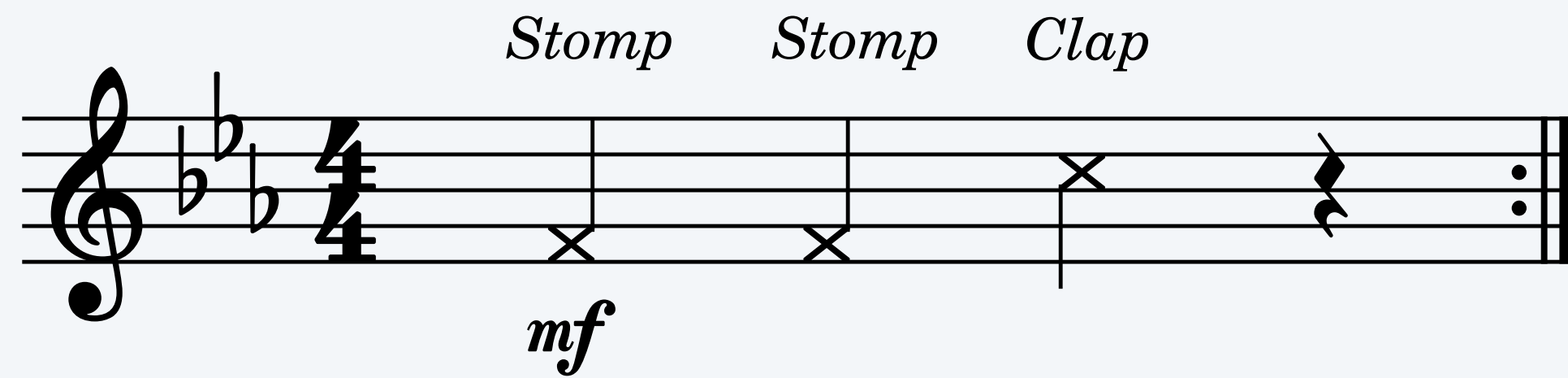


while loop flow chart

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 = 1;
        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



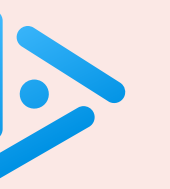
Goal. Repeat a ringtone n times.

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 = 1;  
        while (i <= n) {  
            StdAudio.play(filename);  
            i++;  
        }  
    }  
}
```

	<i>filename</i>	<i>n</i>	<i>i</i>	
	"marimba.wav"	3	1	← before loop
	"marimba.wav"	3	2	
	"marimba.wav"	3	3	
	"marimba.wav"	3	4	← after loop

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



What does the following program do when n is 10?

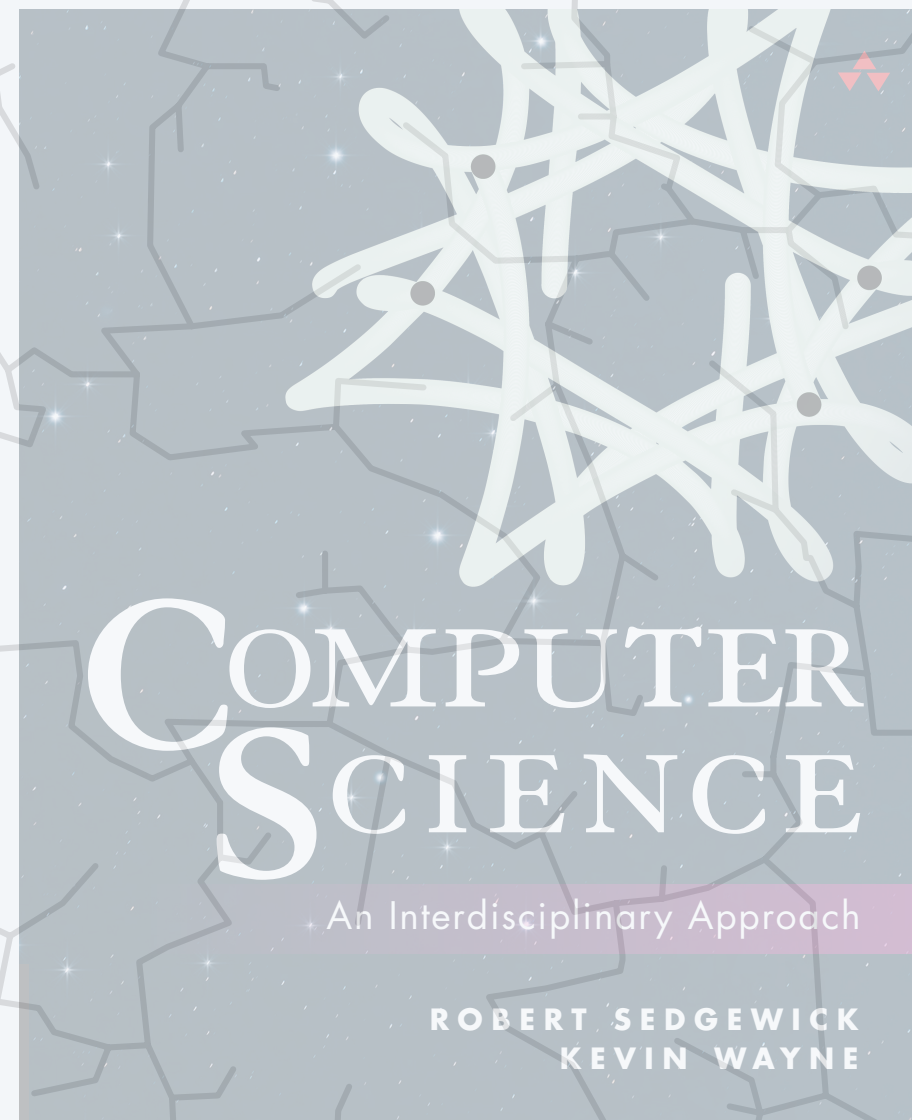
- A. Print 0 to 10.
- B. Print powers of 2, from 2^0 to 2^9 .
- C. Print powers of 2, from 2^0 to 2^{10} .
- D. Print powers of 2, from 2^0 to 2^{11} .
- E. Print powers of 2, from 2^1 to 2^{10} .

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

Examples of *while* loops

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

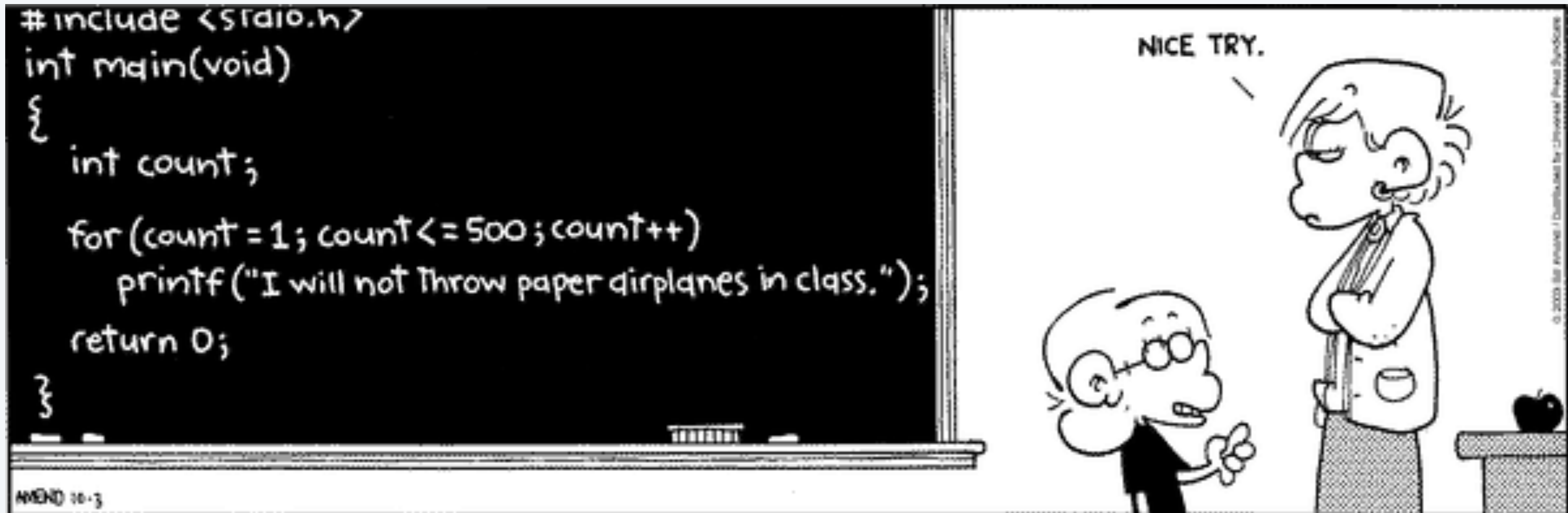


<https://introcs.cs.princeton.edu>

LOOPS

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

A for loop (in C)



Copyright 2004, FoxTrot by Bill Amend

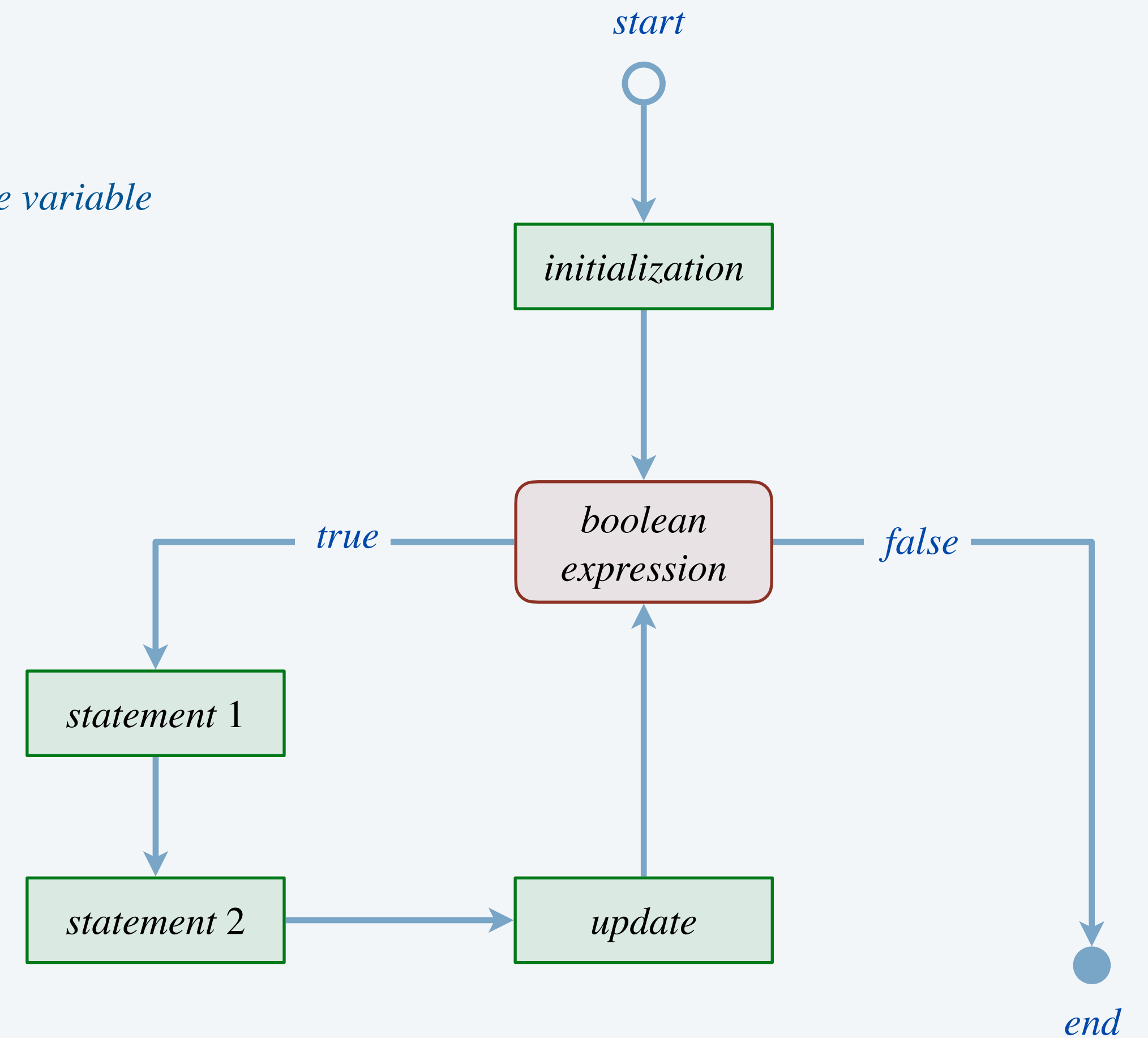
The *for* loop

An alternative repetition structure.

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

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

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 = 1; 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 = 1; i <= n; i++) { product = product * i; }</pre>
<i>print integers</i> <i>from n down to 1</i>	<pre>for (int i = n; i >= 1; i--) { System.out.println(i); }</pre>
<i>infinite loop</i>	<pre>for (;;) { StdAudio.play("heartbeat.wav"); }</pre>

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

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



Q. Which value does the following program print when n is 3?

- A.** 0 1 2 3 2 1 0
- B.** 0 1 0 2 0 1 0
- C.** 0 1 0 2 0 1 0 3
- D.** 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0

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

While loop vs. for loop

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 = 1;
while (i <= n) {
    StdAudio.play(filename);
    i++;
}
```

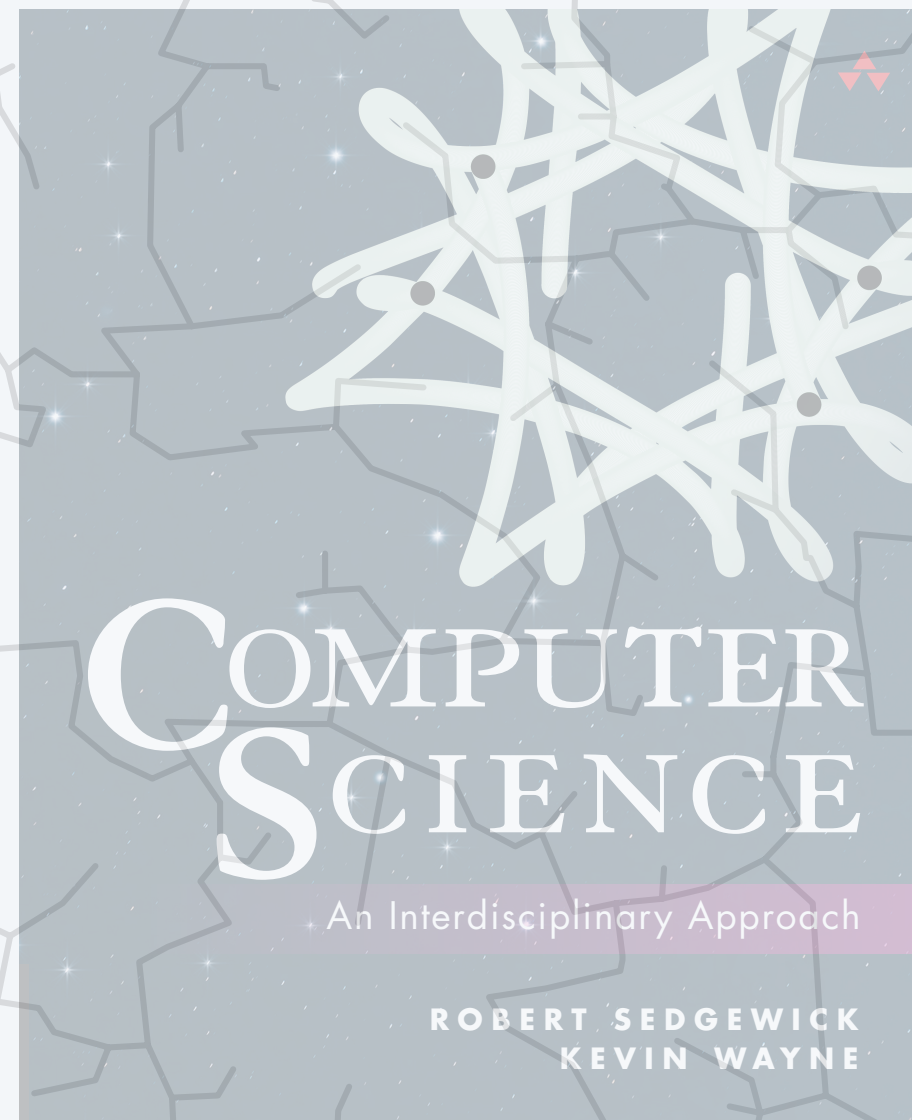
while loop

```
for (int i = 1; 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*
- ▶ *for loops*
- ▶ *nested loops*
- ▶ *image processing*





Suppose $m = 10$ and $n = 5$. How many lines of output does the following program produce?

- A. 10
- B. 15
- C. 50
- D. 55
- E. 60

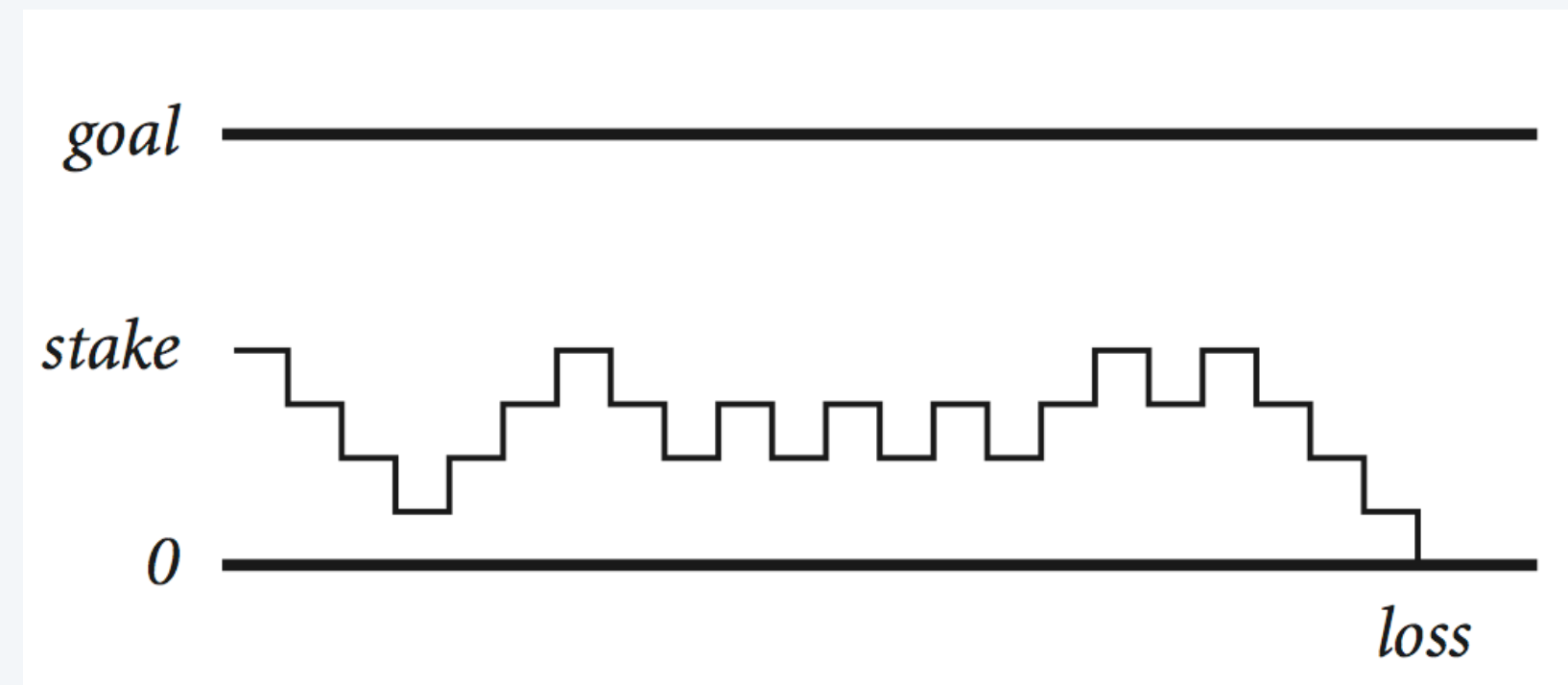
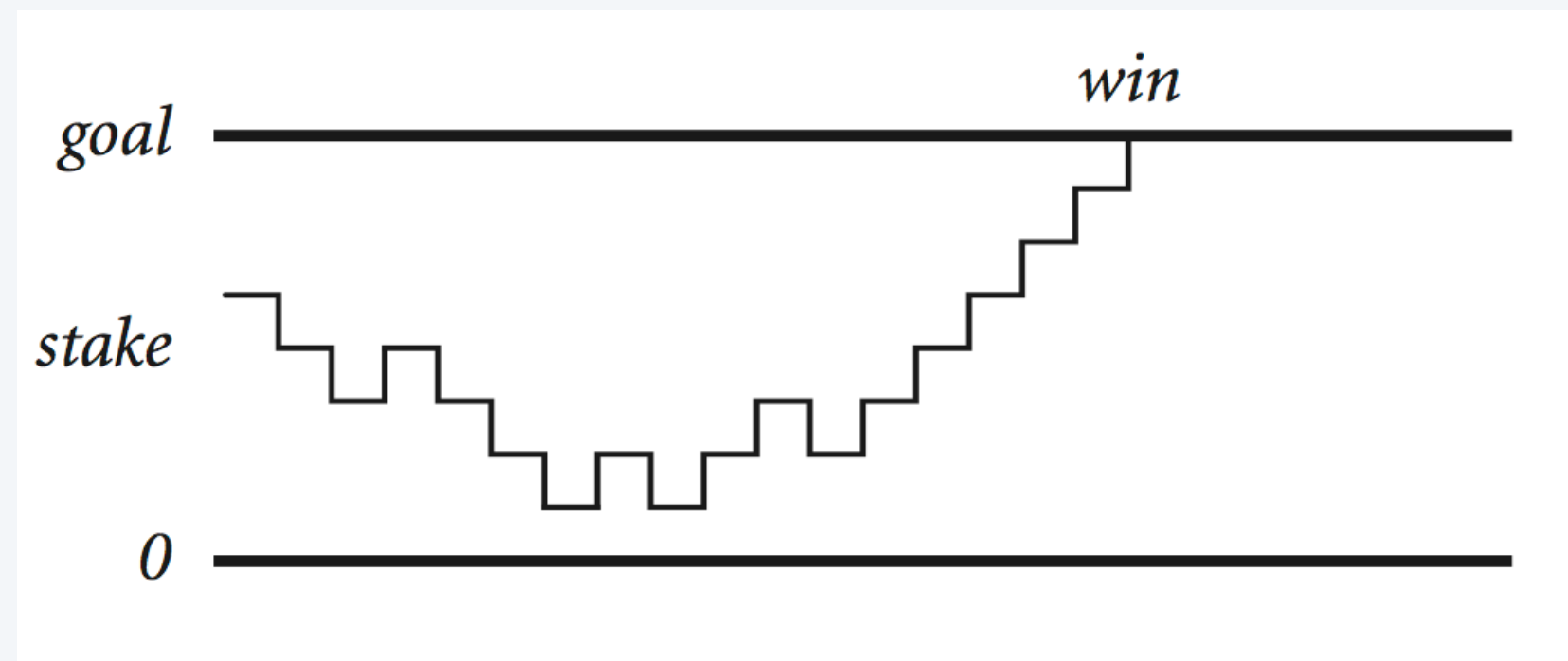
```
public class Mystery {  
    public static void main(String[] args) {  
        int m = Integer.parseInt(args[0]);  
        int n = Integer.parseInt(args[1]);  
  
        for (int i = 1; i <= m; i++) {  
            for (int j = 1; j <= n; j++) {  
                System.out.println(i + "-" + j);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

for loop nested within a for loop

Gambler's ruin problem

Gambler's ruin. A gambler starts with $\$stake$ and places $\$1$ fair bets.

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



Q1. What are the chances of winning?


Q2. How many bets until win or loss?

One approach. [Monte Carlo simulation]

- Perform one experiment using simulated coin flips.
- Repeat experiment many times and collect statistics.



Gambler's ruin problem: one experiment

Warmup. Simulate one experiment.  *use while loop*
(don't know how many iterations)

```
public class GamblerWarmup {  
    public static void main(String[] args) {  
        int stake = Integer.parseInt(args[0]);  
        int goal = Integer.parseInt(args[1]);  
  
        int cash = stake;  
        System.out.println(cash);  
        while ((cash > 0) && (cash < goal)) {  
            if (Math.random() < 0.5) cash++;  
            else cash--;  
            System.out.println(cash);  
        }  
    }  
}
```

print trace
(for debugging only)

 *if-else statement nested*
within a while loop

```
~/loops> java GamblerWarmup 4 10  
4  
5  
4  
3  
4  
3  
2  
1  
2  
1  
0
```


Monte Carlo simulation of gambler's ruin problem

```
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 = 1; 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);
    }
}
```

do trials experiments

initialize cash to stake for each experiment

do one experiment

make one bet

*if goal met in experiment t,
record as a win*

```
~/cos126/loops> java Gambler 5 25 1000
191 wins of 1000
```

```
~/cos126/loops> java Gambler 5 25 1000
183 wins of 1000
```

Digression: simulation vs. mathematical analysis

Facts. [known via probability theory]

- Probability of winning = $stake \div goal$.
- Expected number of bets = $stake \times (goal - stake)$.

Ex. [$stake = 500$, $goal = 2500$]

- 20% chance of winning.
- Expect to make 1 million bets per experiment.

Remarks.

- For gambler's ruin, mathematical analysis is well known.
- Computer simulation agrees with math.
- For more complicated variants, math may be beyond reach.
- Monte Carlo simulations widely used in STEM.

stake *goal* *trials*
↓ ↓ ↓

```
~/cos126/loops> java Gambler 500 2500 1000  
197 wins of 1000  
~/cos126/loops> java Gambler 500 2500 1000  
202 wins of 1000
```

*takes about 15 seconds
(makes about 1 billion bets)*

Integer factorization

Goal. Given a positive integer n , find its prime factorization.

$$98 = 2 \times 7 \times 7$$

$$3,757,208 = 2 \times 2 \times 2 \times 7 \times 13 \times 13 \times 397$$

$$11,111,111,111,111,111 = 2,071,723 \times 5,363,222,357$$

Grade-school factoring algorithm.

FACTOR(n)

Consider each potential divisor d between 2 and n :

- *while* d is a divisor of n :
 - *print* d
 - $n \leftarrow n / d$
-

Critical application. Cryptography.



← *security of internet commerce relies on
difficulty of factoring very large integers*

Integer factorization

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

*try all possible
divisors d*

*if d is a divisor,
factor it out*

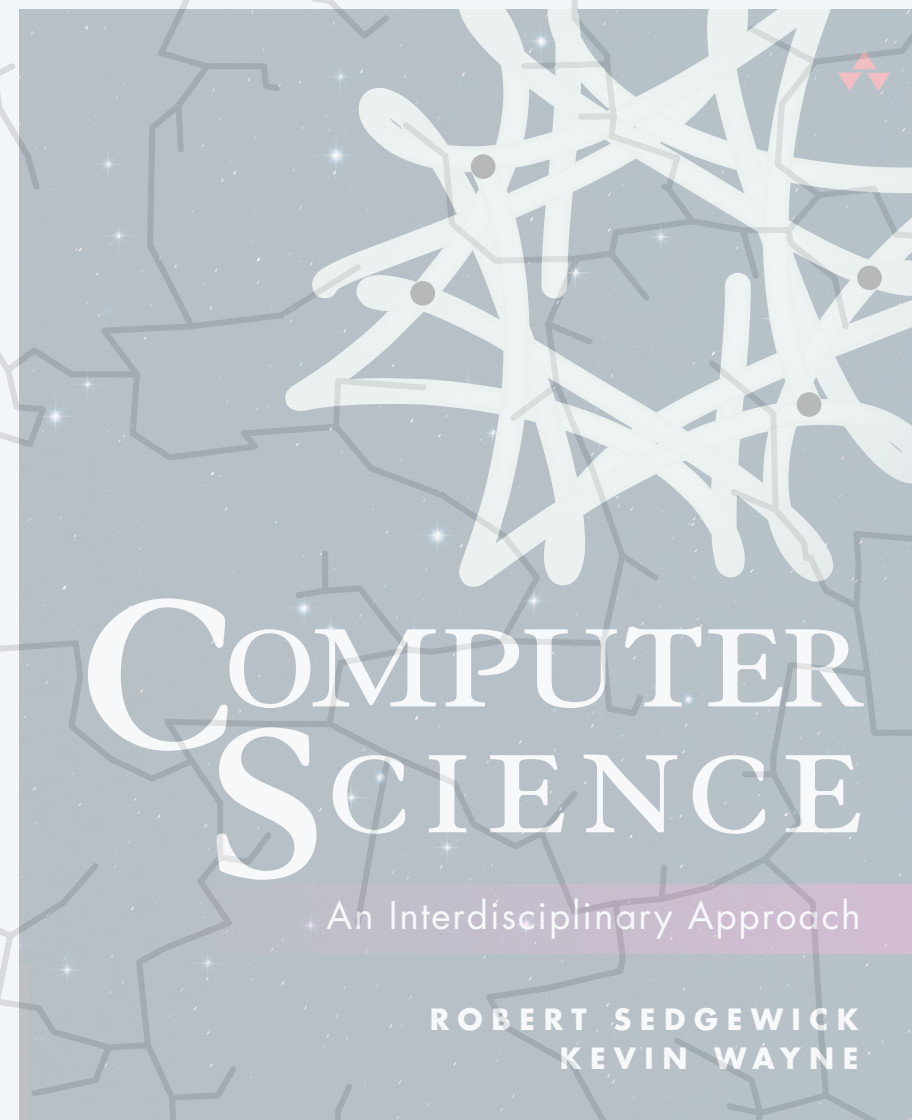
```
~/cos126/loops> java Factors 98  
2 7 7  
  
~/cos126/loops> java Factors 3757208  
2 2 2 7 13 13 397  
  
~/cos126/loops> java Factors 97  
97  
  
~/cos126/loops> java Factors 11111111111111111111  
2071723 5363222357
```

takes a few seconds

Remark 1. Uses *long* instead of *int* to support integers between -2^{63} and $2^{63} - 1$.

Remark 2. Way too **slow** to break cryptography.

*can be sped up substantially by stopping when $d > \sqrt{n}$
(but still way too slow)*



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

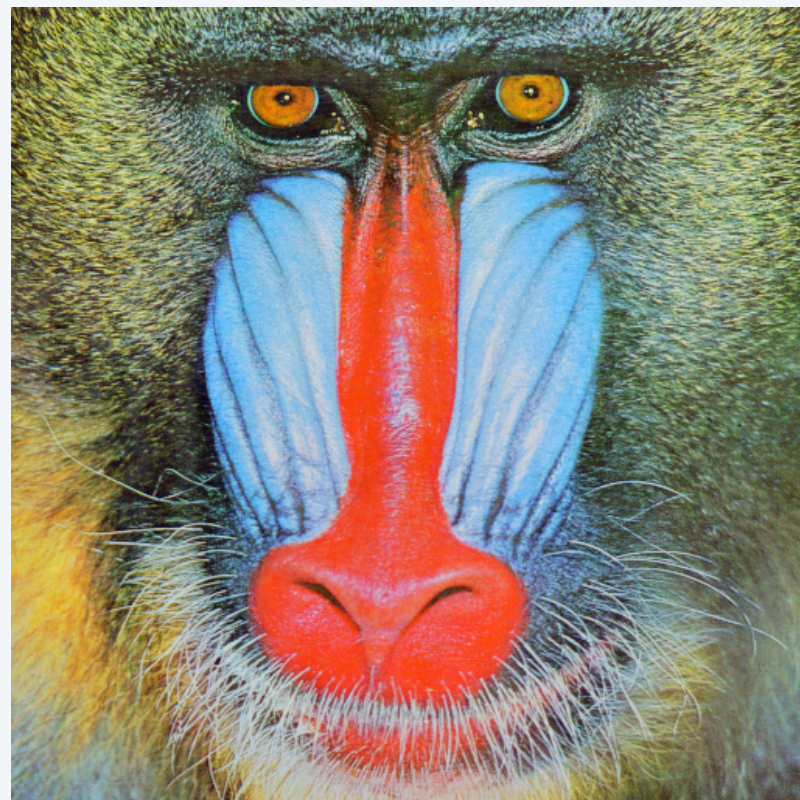
LOOPS

- ▶ *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

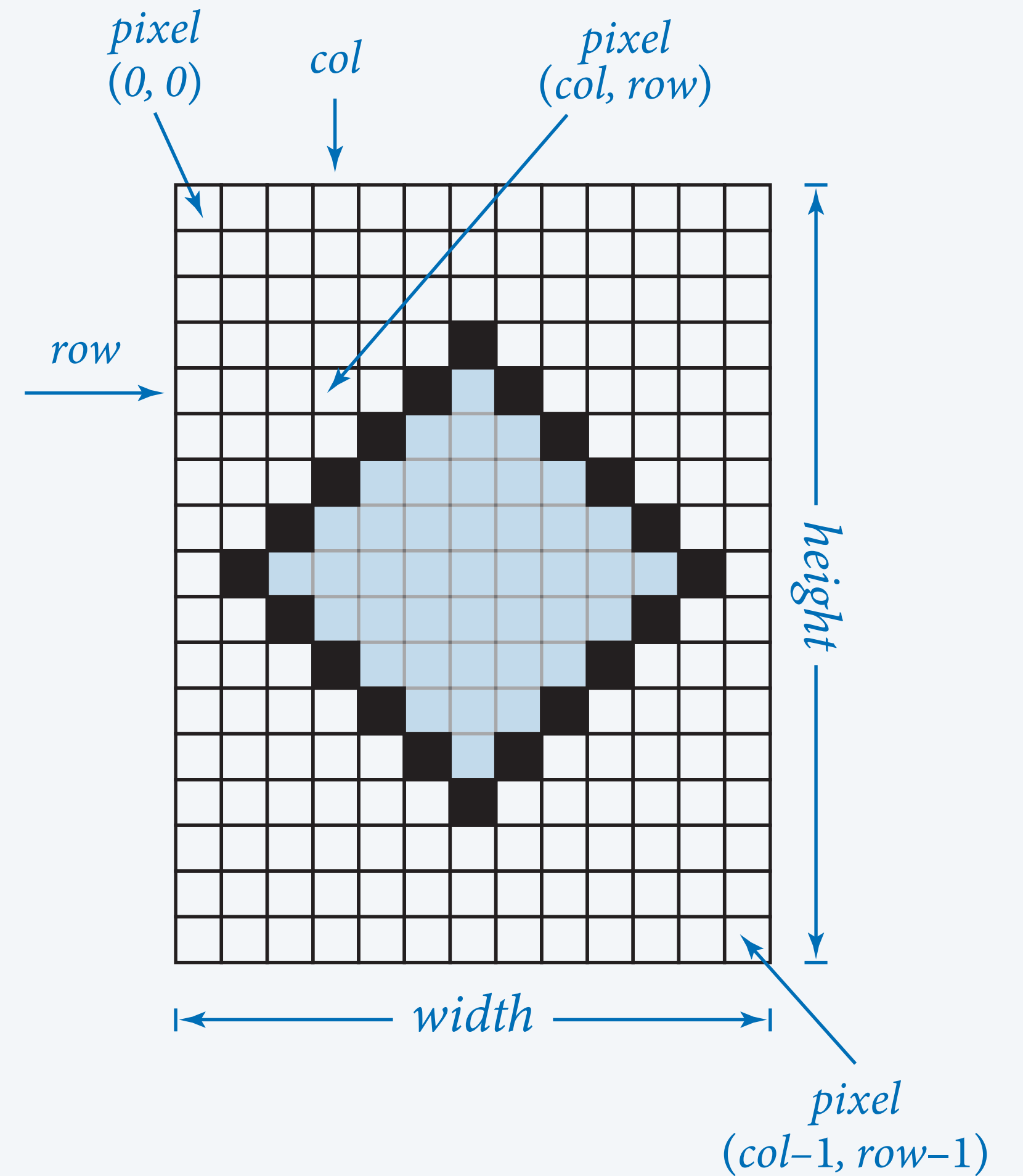


Image-processing conventions.

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

← warning: different conventions from matrices and Cartesian coordinates

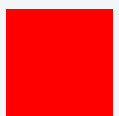
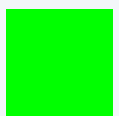


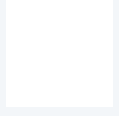
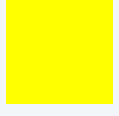
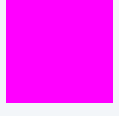
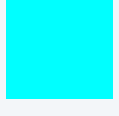

RGB color model

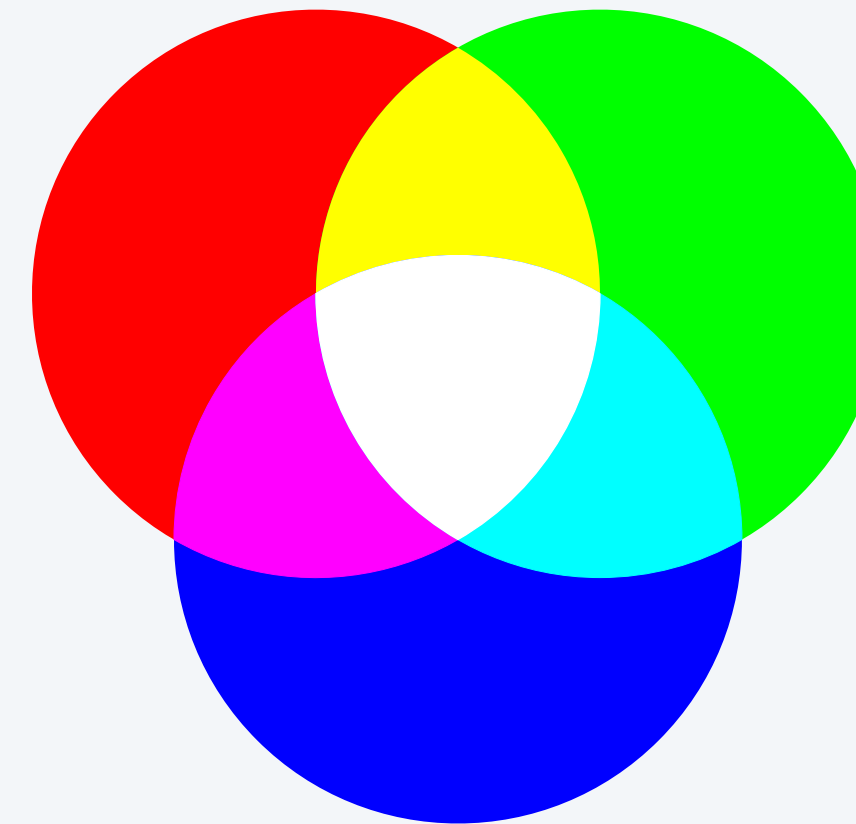
Color is a sensation in the eye from electromagnetic radiation.

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.



name	red	green	blue	color
<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

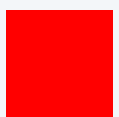
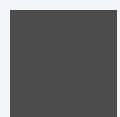
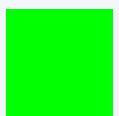
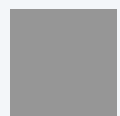




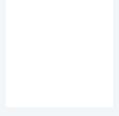
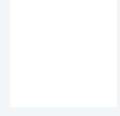
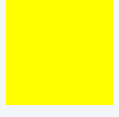
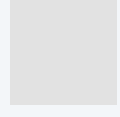
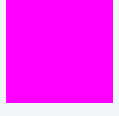
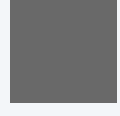
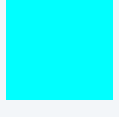
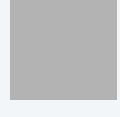

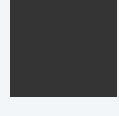
Goal. Convert color image to grayscale.

*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:

$$Y = 0.299 R + 0.587 G + 0.114 B$$



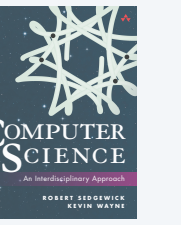
name	red	green	blue	color	lum	gray
<i>red</i>	255	0	0		76	
<i>green</i>	0	255	0		150	
<i>blue</i>	0	0	255		29	
<i>black</i>	0	0	0		0	
<i>white</i>	255	255	255		255	
<i>yellow</i>	255	255	0		226	
<i>magenta</i>	255	0	255		105	
<i>cyan</i>	0	255	255		179	
<i>book blue</i>	0	64	128		52	

$$Y = 0.299 R + 0.587 G + 0.114 B$$

$$= 0.299 (0) + 0.587 (64) + 0.114 (128)$$

$$= 52.16$$

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

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

```
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)

```
⋮
```

```
⋮
```


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

Image processing: color image filters



original



grayscale



sepia



duotone



brighter



darker



RGB layers



negative

Image processing: shape masks



original



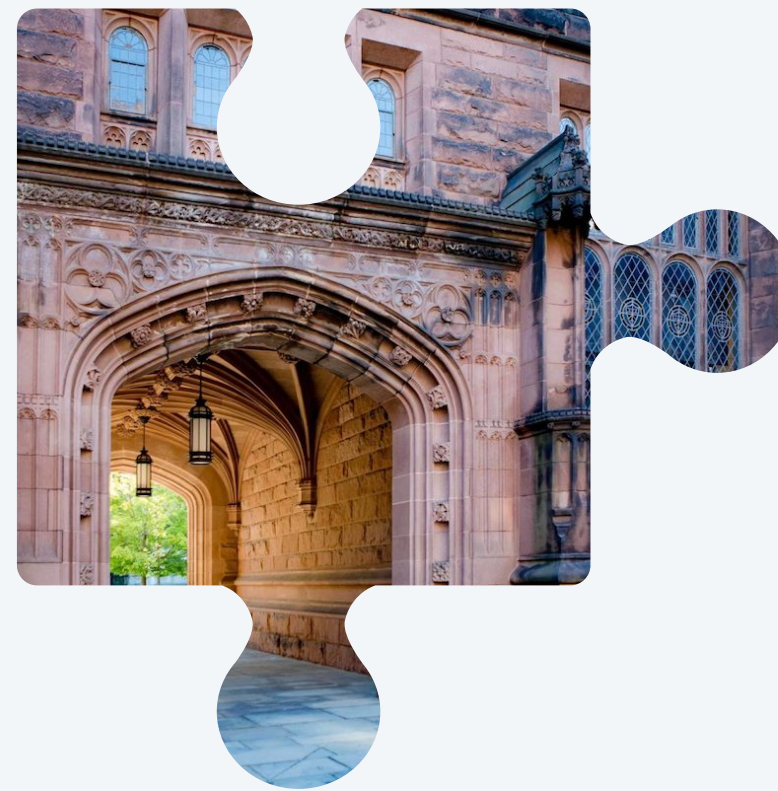
rounded rectangle



oval



heart



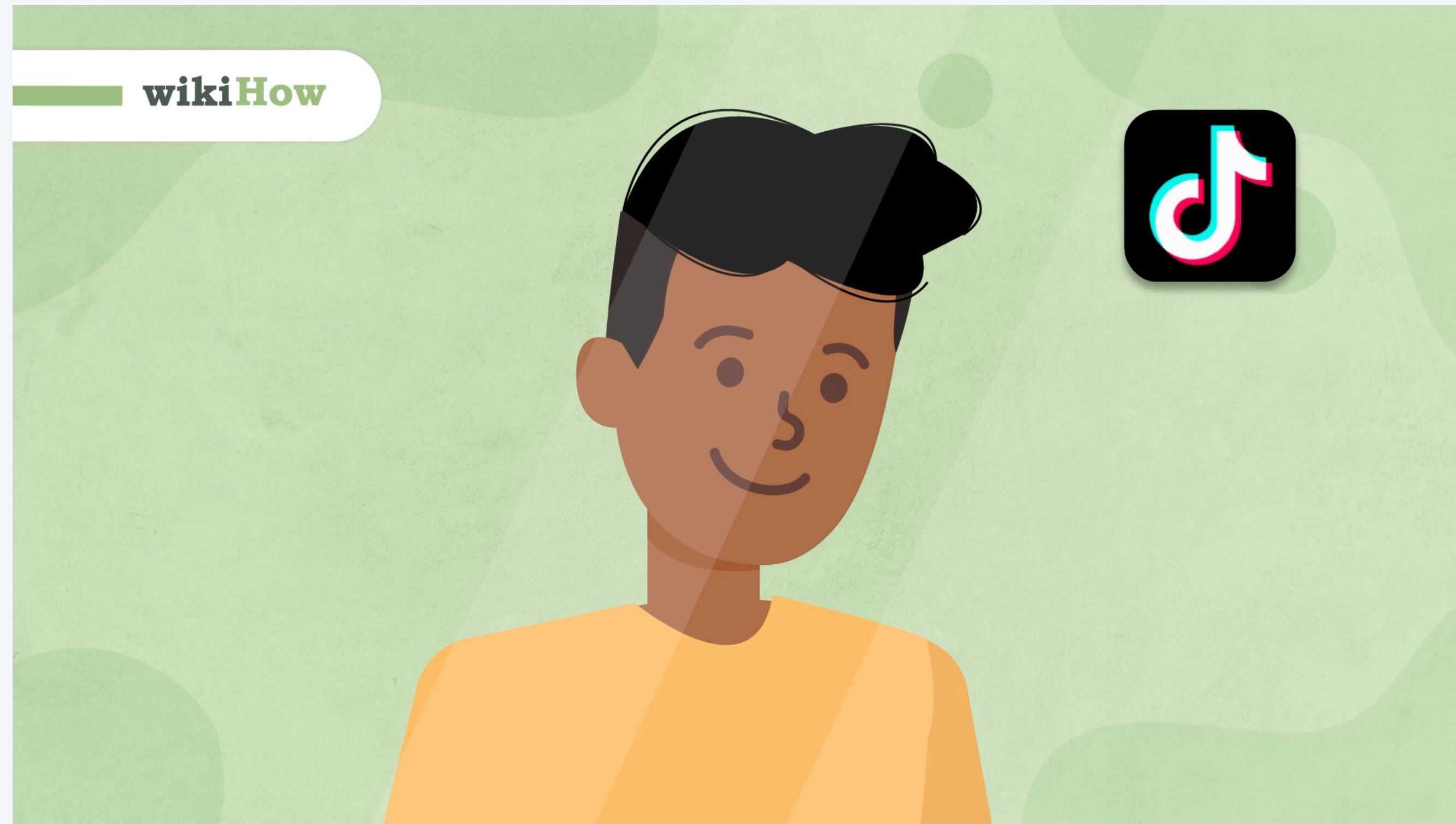
puzzle piece



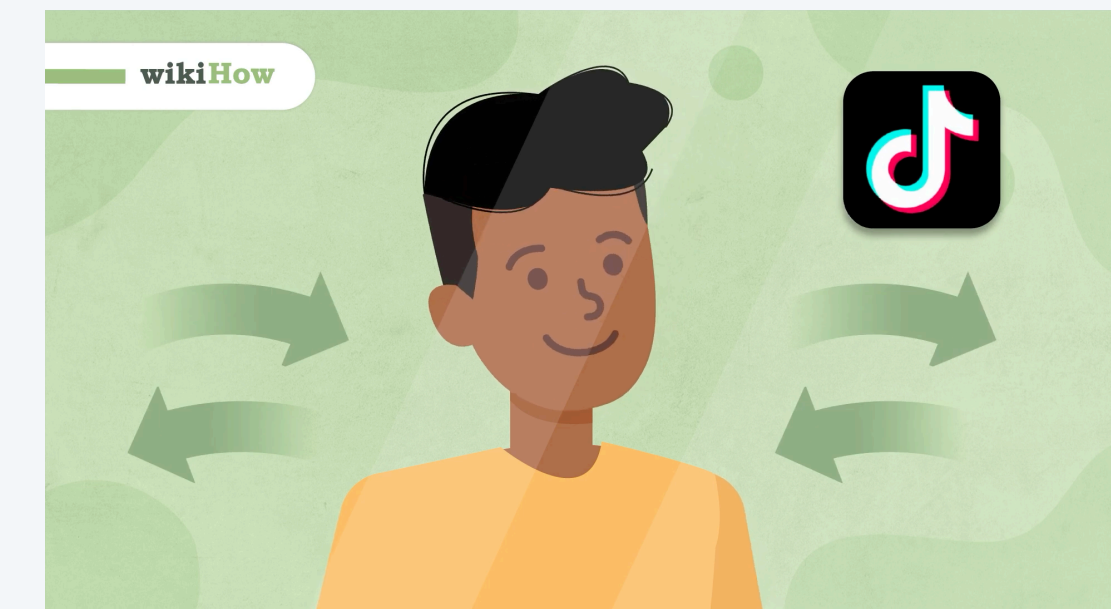
tiger

Flip an image horizontally (inverted filter)

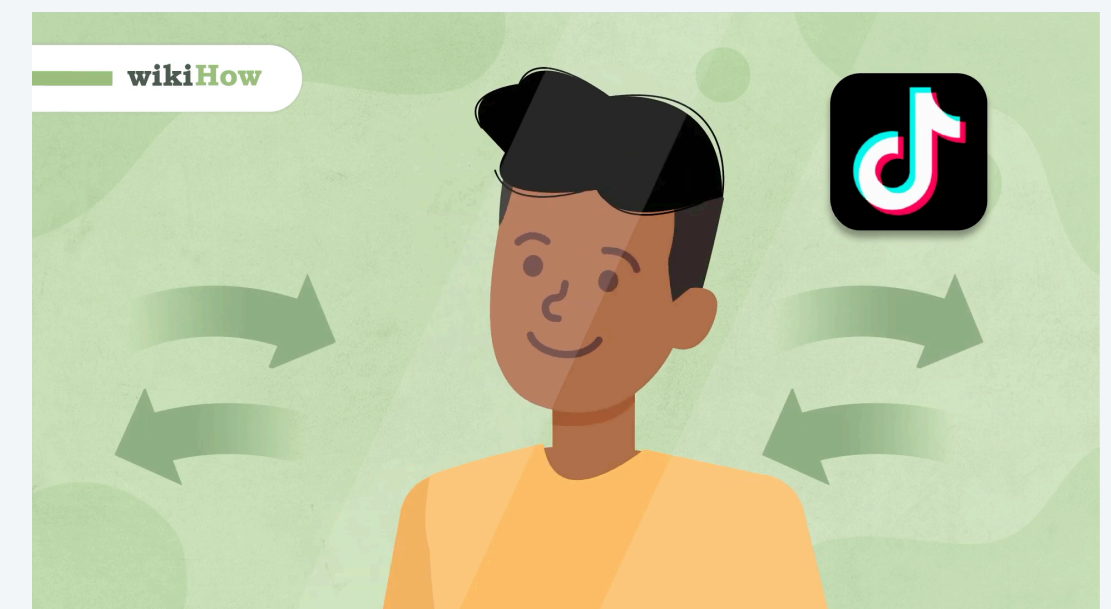
Goal. Flip an image horizontally, like looking into a mirror. ← on Zoom, Instagram, TikTok, ...



<https://www.wikihow.com/Inverted-Filter>



original



Tik Tok inverted filter

Deja Vu challenge

Deja Vu challenge. Record video of face while repeatedly turning on/off inverted filter.

Disclaimer. COS 126 is not liable for damage to self-esteem.



VICE

I Tried The Inverted Filter on TikTok to Find Out Why It's Messing With People's Self-Esteem

TikTokers are having meltdowns seeing their faces inverted. I tried the cursed filter out myself.



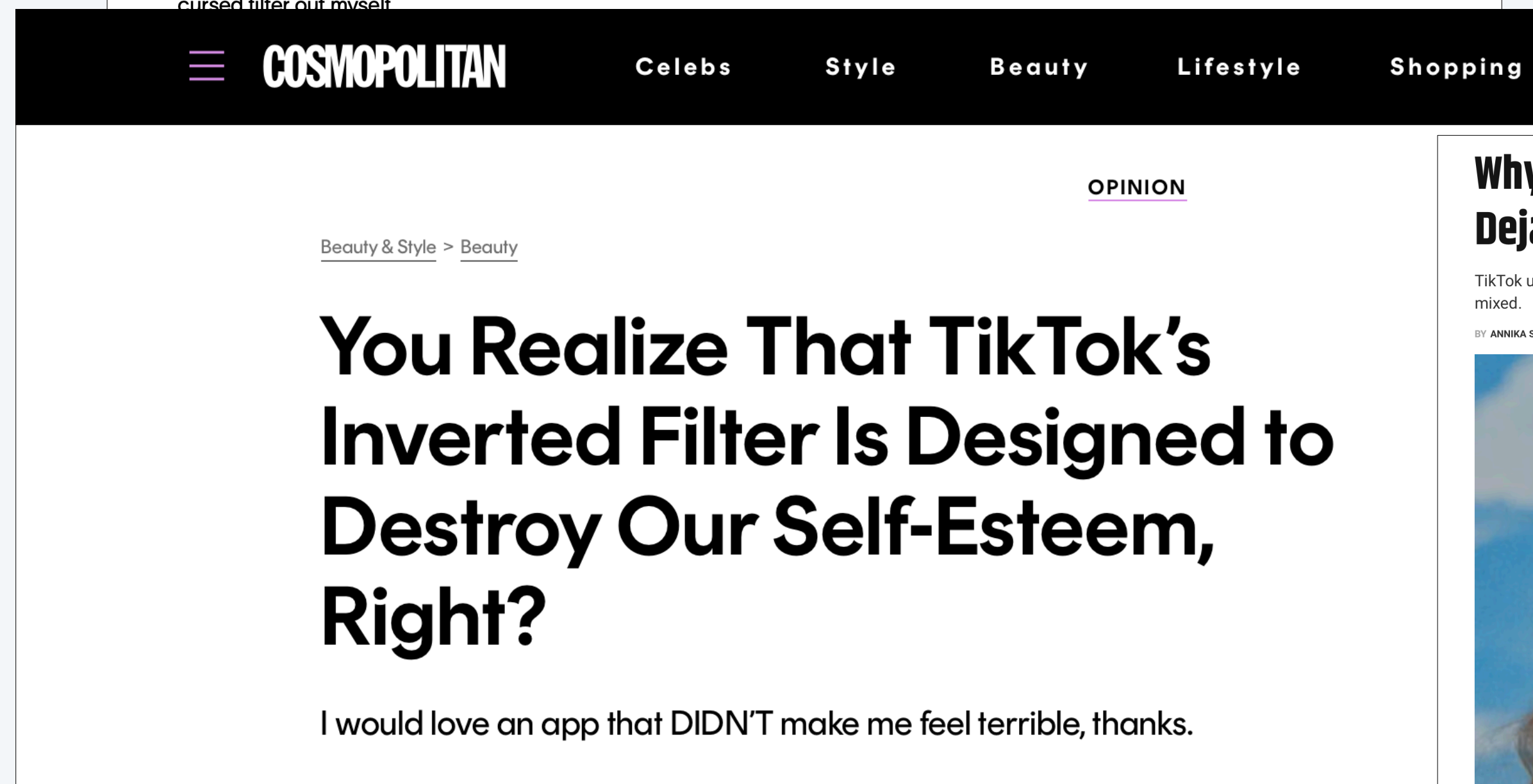
CNET [Join/Login](#)

Tech Money Home Wellness Home Internet Energy Deals Sleep Price Finder More

Culture > Internet Culture

Why TikTok's inverted filter is turning people's self-esteem upside down

Commentary: The Deja Vu challenge is all fun and games until it warps your self-image.



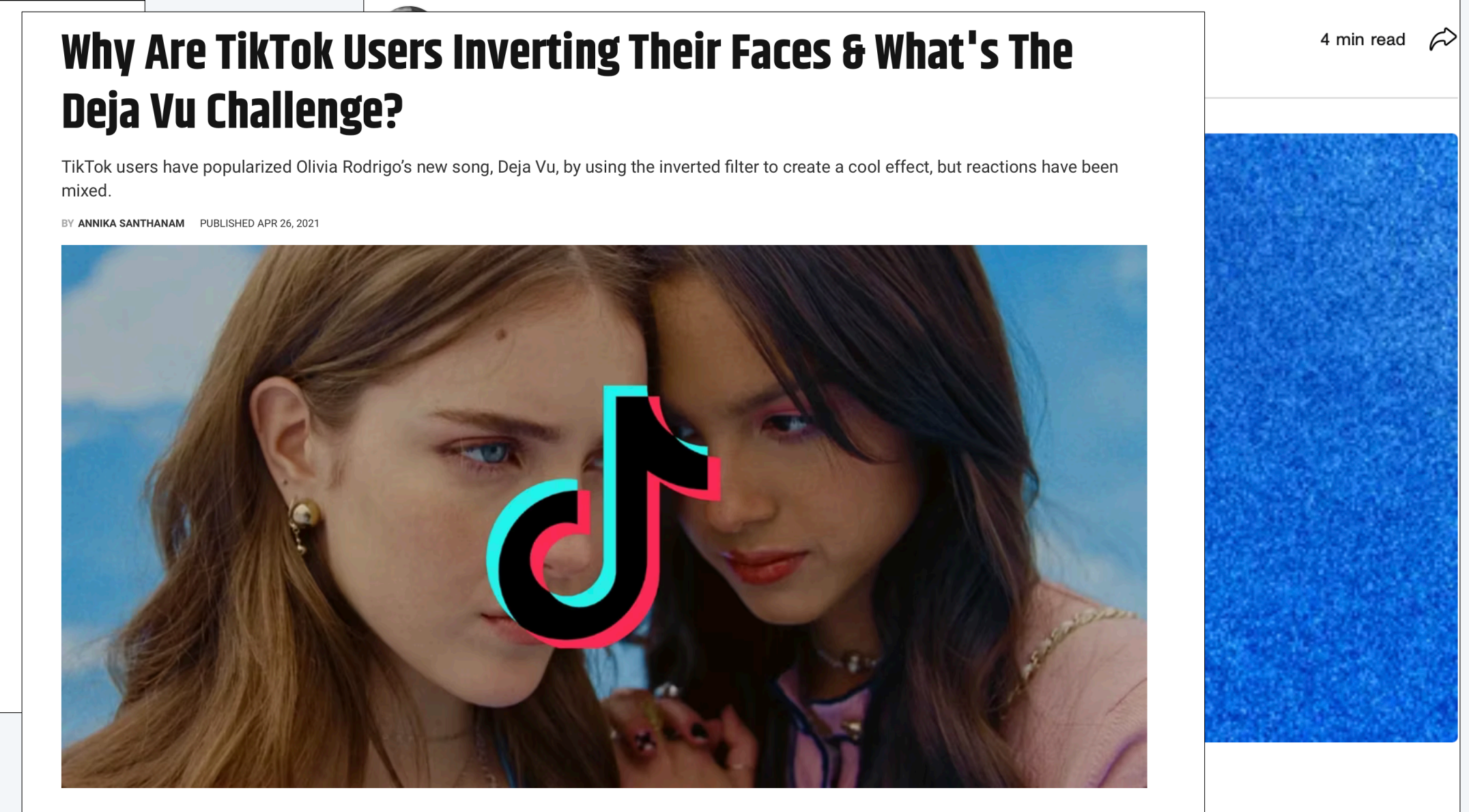
COSMOPOLITAN [Celebs](#) [Style](#) [Beauty](#) [Lifestyle](#) [Shopping](#)

OPINION

Beauty & Style > Beauty

You Realize That TikTok's Inverted Filter Is Designed to Destroy Our Self-Esteem, Right?


I would love an app that DIDN'T make me feel terrible, thanks.



Why Are TikTok Users Inverting Their Faces & What's The Deja Vu Challenge?

TikTok users have popularized Olivia Rodrigo's new song, Deja Vu, by using the inverted filter to create a cool effect, but reactions have been mixed.

BY ANNIKA SANTHANAM PUBLISHED APR 26, 2021



4 min read

Flip an image horizontally: demo



Goal. Flip an image horizontally, like looking into a mirror.

(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)

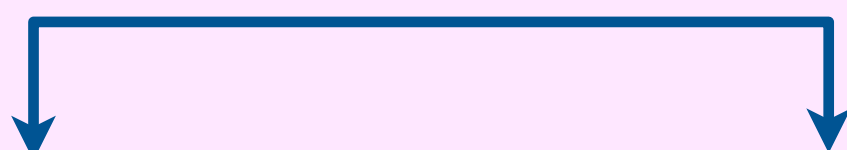
original image

Flip an image horizontally: demo



Goal. Flip an image horizontally, like looking into a mirror.

Algorithm. For each pixel (col, row) , swap with pixel $(width - col - 1, row)$.



(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

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

flipped image

Flip an image horizontally: implementation

Goal. Flip an image horizontally, like looking into a mirror.

Algorithm. For each pixel (col, row) , swap with pixel $(width - col - 1, row)$.

```
for (int col = 0; col < width / 2; col++) {  
    for (int row = 0; row < height; row++) {  
        int r1 = StdPicture.getRed(col, row);  
        int g1 = StdPicture.getGreen(col, row);  
        int b1 = StdPicture.getBlue(col, row);  
        int r2 = StdPicture.getRed(width - col - 1, row);  
        int g2 = StdPicture.getGreen(width - col - 1, row);  
        int b2 = StdPicture.getBlue(width - col - 1, row);  
        StdPicture.setRGB(col, row, r2, g2, b2);  
        StdPicture.setRGB(width - col - 1, row, r1, g1, b1);  
    }  
}  
StdPicture.show();
```

why not width ?

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





What image does the following code fragment produce?

- A. Original image.
- B. Horizontal flip.
- C. Vertical flip.

```
for (int row = 0; row < height; row++) {  
    for (int col = 0; col < width / 2; col++) {  
        int r1 = StdPicture.getRed(col, row);  
        int g1 = StdPicture.getGreen(col, row);  
        int b1 = StdPicture.getBlue(col, row);  
        int r2 = StdPicture.getRed(width - col - 1, row);  
        int g2 = StdPicture.getGreen(width - col - 1, row);  
        int b2 = StdPicture.getBlue(width - col - 1, row);  
        StdPicture.setRGB(col, row, r2, g2, b2);  
        StdPicture.setRGB(width - col - 1, row, r1, g1, b1);  
    }  
}  
StdPicture.show();
```

*switched order of
two for loops*

Live coding (Deja Vu challenge)



```
public class DejaVuChallenge {
    public static void main(String[] args) {
        // read file
        String filename = args[0];
        StdPicture.read(filename);

        // get dimensions
        int width = StdPicture.width();
        int height = StdPicture.height();

        while (true) {
            // flip image horizontally
            for (int col = 0; col < width / 2; col++) {
                for (int row = 0; row < height; row++) {
                    int r1 = StdPicture.getRed(col, row);
                    int g1 = StdPicture.getGreen(col, row);
                    int b1 = StdPicture.getBlue(col, row);
                    int r2 = StdPicture.getRed(width - col - 1, row);
                    int g2 = StdPicture.getGreen(width - col - 1, row);
                    int b2 = StdPicture.getBlue(width - col - 1, row);
                    StdPicture.setRGB(col, row, r2, g2, b2);
                    StdPicture.setRGB(width - col - 1, row, r1, g1, b1);
                }
            }

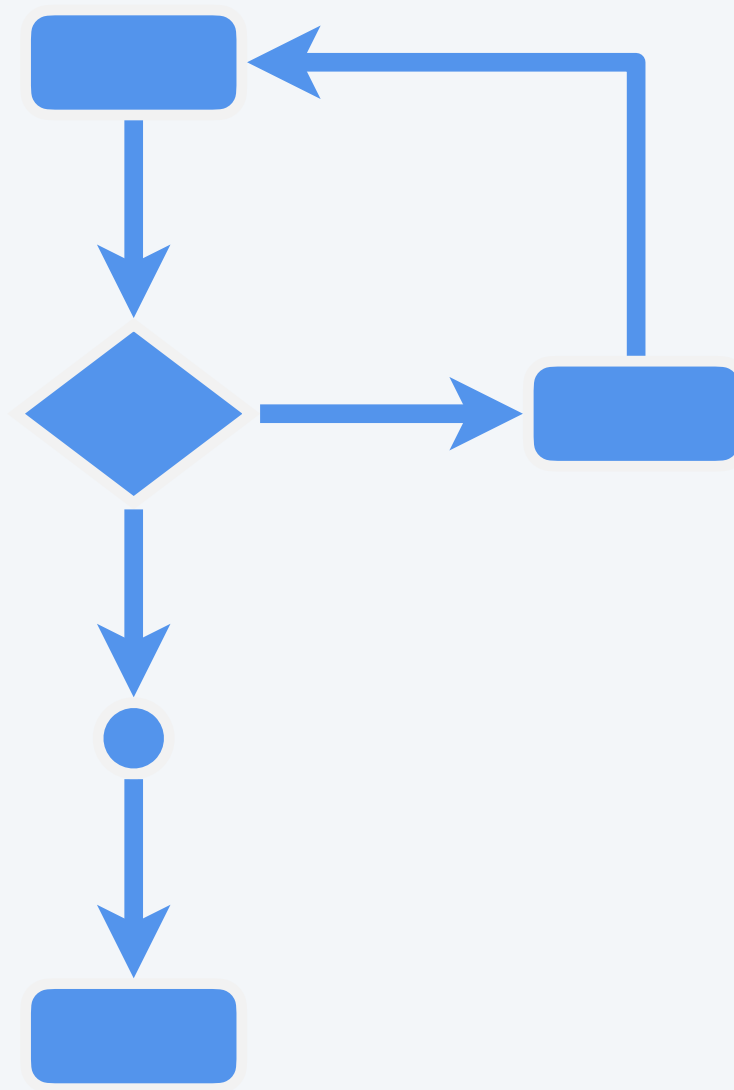
            // display image and pause
            StdPicture.show();
            StdPicture.pause(100);
        }
    }
}
```

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 a color.



control flow with loops

Credits

media	source	license
<i>Buzz Lightyear</i>	alphacoders.com	
<i>Rainbow Infinity</i>	Adobe Stock	education license
<i>Stomp–Stomp–Clap</i>	Queen	
<i>Ringtone Icon</i>	Wikimedia	public domain
<i>Marimba Ringtone</i>	Apple iPhone	
<i>Sonar Ringtone</i>	Apple iPhone	
<i>Coin Toss</i>	clipground.com	CC BY 4.0
<i>Paper Airplanes</i>	FoxTrot by Bill Amend	
<i>Heartbeat</i>	freesound.org	CC BY 4.0
<i>Amen Break</i>	The Winstons	

Credits

media	source	license
<i>Russian Nesting Dolls</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Cryptography Icon</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Image Processing Icon</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Mandrill</i>	<u>USC SIPI Image Database</u>	
<i>Johnson Arch</i>	<u>Danielle Alio Capparella</u>	by photographer
<i>RGB Color Model</i>	<u>Wikimedia</u>	<u>Kopimi</u>
<i>LGBTQ+ Eye</i>	<u>Wikimedia</u>	<u>CC BY 2.0</u>
<i>Inverted Filter</i>	<u>WikiHow</u>	