

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

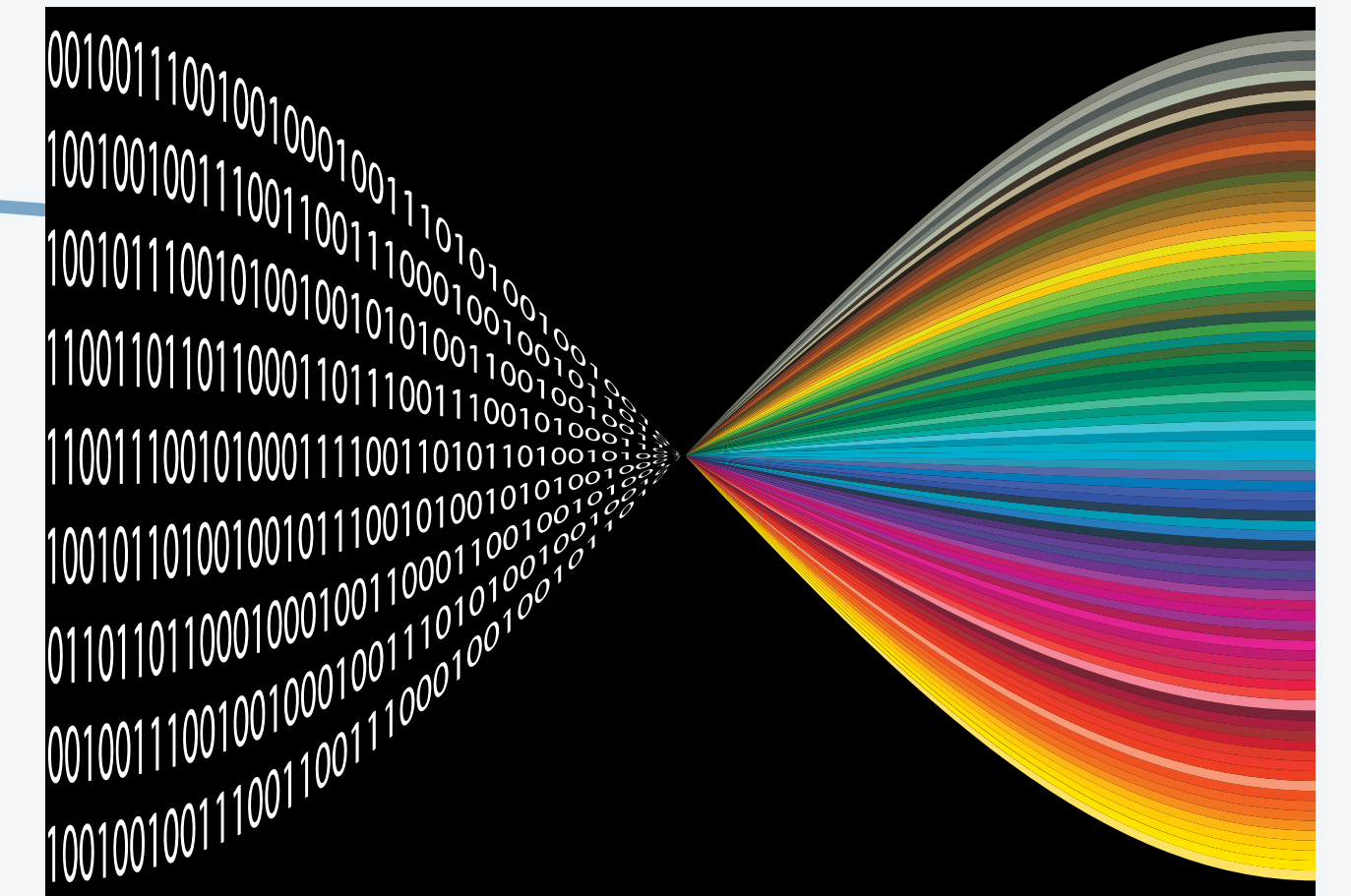
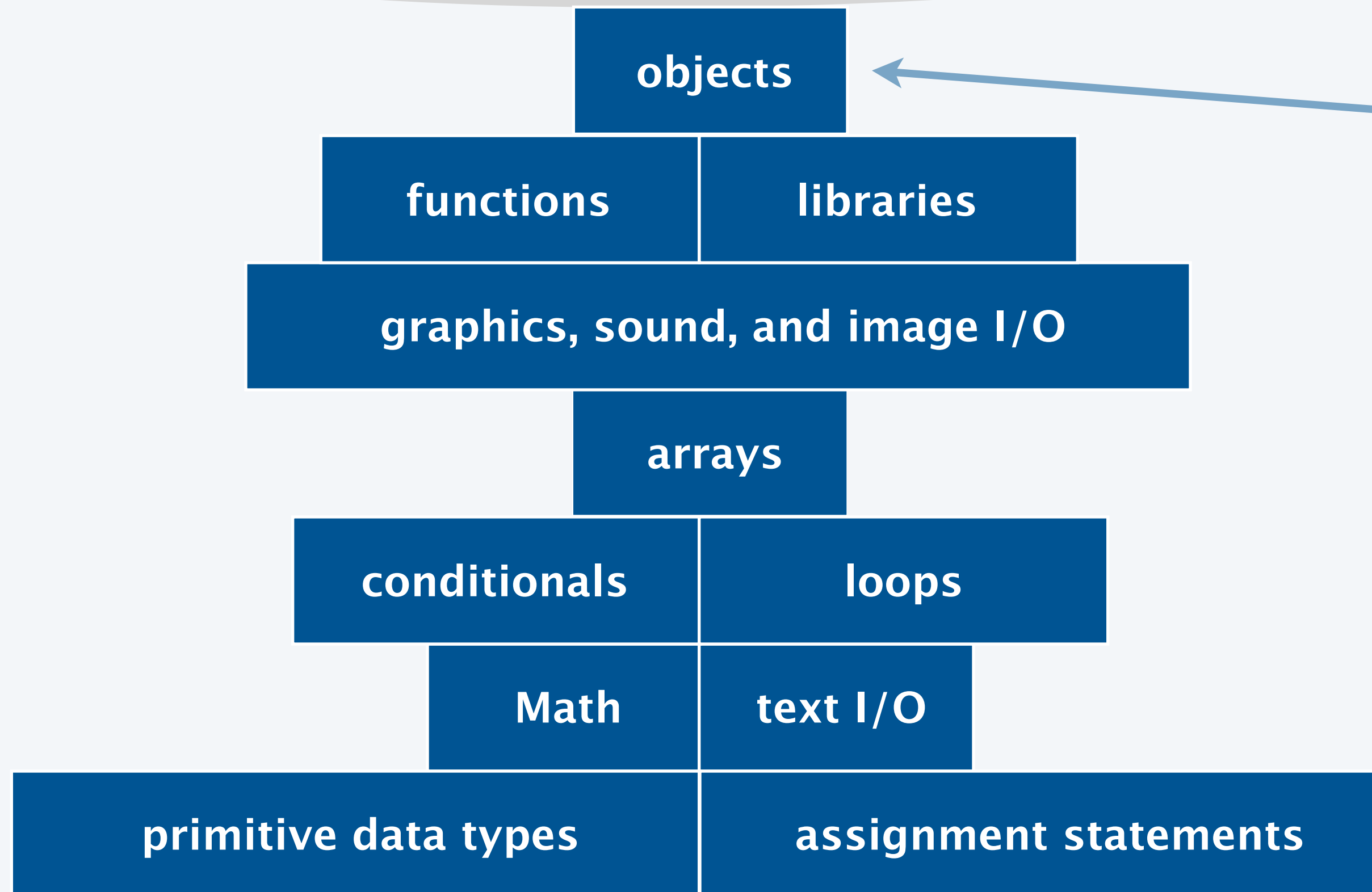
## 3.1 USING DATA TYPES

---

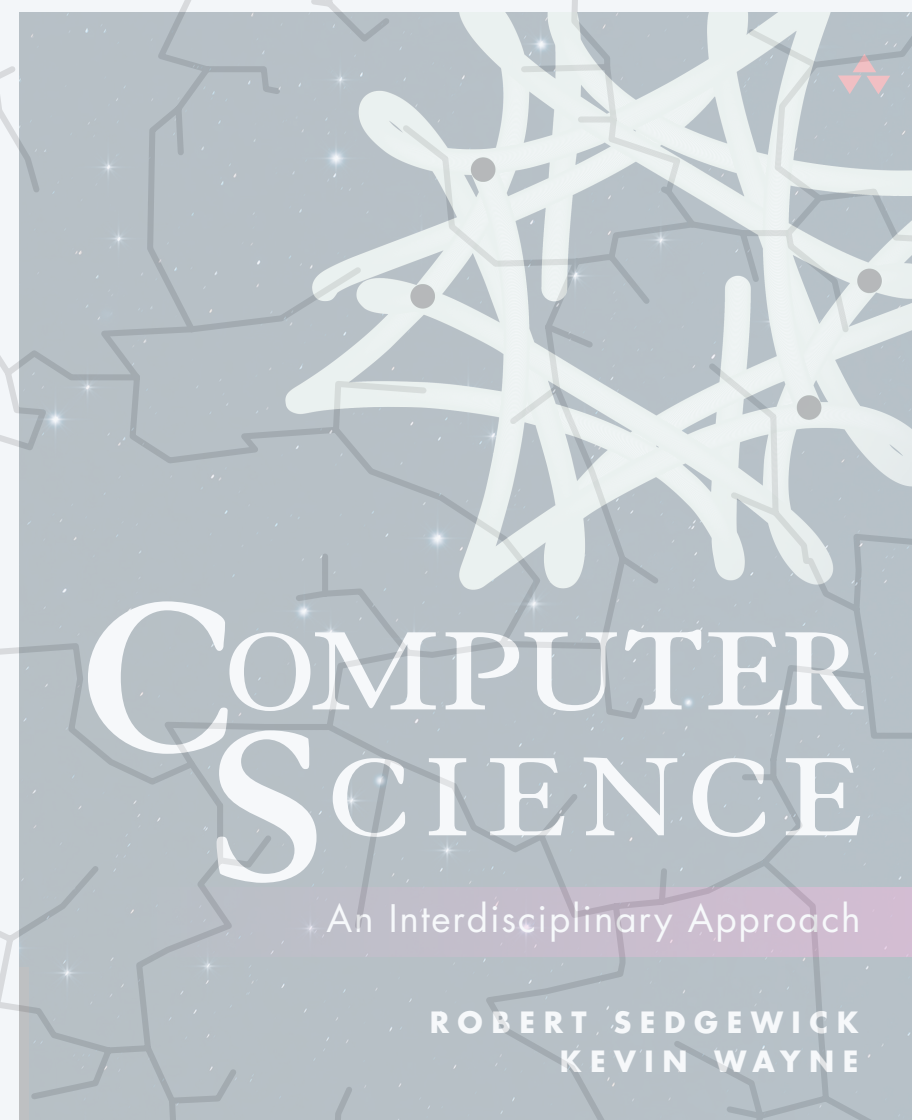
- ▶ *overview*
- ▶ *string processing*
- ▶ *color*
- ▶ *image processing*

# Basic building blocks for programming

any program you might want to write



*use data types that represent strings, colors, pictures, ...*



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

## 3.1 USING DATA TYPES

---

- ▶ *overview*
- ▶ *string processing*
- ▶ *color*
- ▶ *image processing*

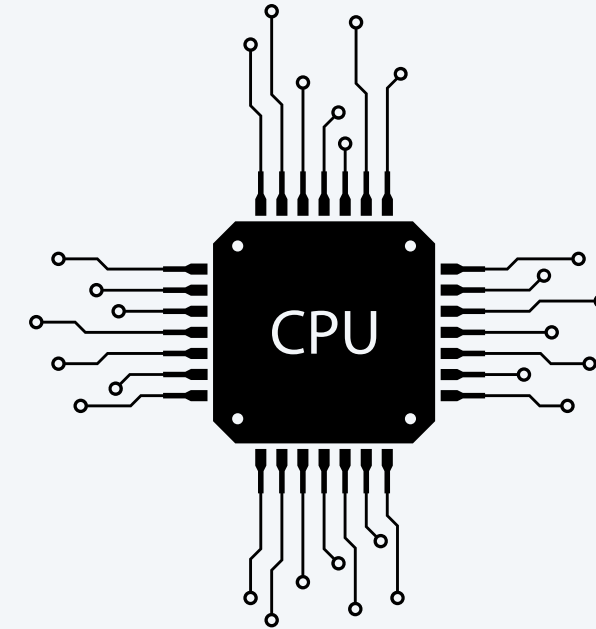
# Primitive data types

---

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

## Primitive types.

- Values map directly to machine representations.
- Operations map directly to machine instructions.




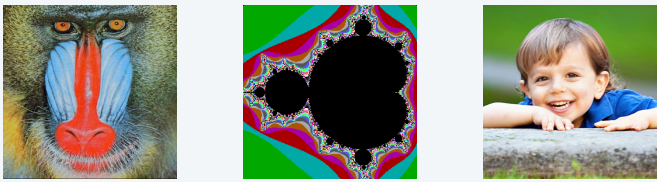
primitive type	set of values	example values	operations
<code>int</code>	<i>integers</i>	17 -12345	add, subtract, multiply, divide, ...
<code>double</code>	<i>floating-point numbers</i>	2.5 -0.125	add, subtract, multiply, divide, ...
<code>boolean</code>	<i>truth values</i>	true false	and, or, not, ...
⋮	⋮	⋮	⋮

# Reference data types

---

**Goal.** Write programs that process other types of data.

- Strings, colors, pictures, ...
- Points, circles, complex numbers, vectors, matrices, ...
- GUIs, database connections, neural networks, plots, ...

reference type	set of values	example values	operations
String	<i>sequences of characters</i>	"Hello, World" "COS 126 is fun"	length, concatenate, compare, extract substring, search, ...
Color	<i>three 8-bit integers</i>		get RGB component, brighter, darker, ...
Picture	<i>2D array of colors</i>		get/set color of pixel, width, height, show, save, ...
⋮	⋮	⋮	⋮

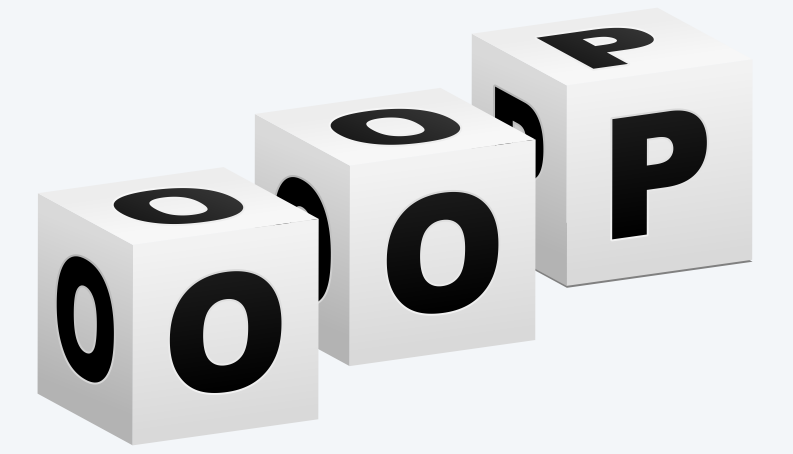
# Object-oriented programming (OOP)

---

**Goal.** Write programs that process other types of data.

- Strings, colors, pictures, ...
- Points, circles, complex numbers, vectors, matrices, ...
- GUIs, database connections, neural networks, plots, ...

*OOP empowers you  
to do this (and more!)*



**This lecture.** Use pre-existing data types.

**Next lecture.** Create your own data types.



Which reference data types have we encountered in this course so far?

- A. Arrays.
- B. Strings.
- C. Both A and B.
- D. Neither A nor B.

# Using a reference data type: constructors and instance methods

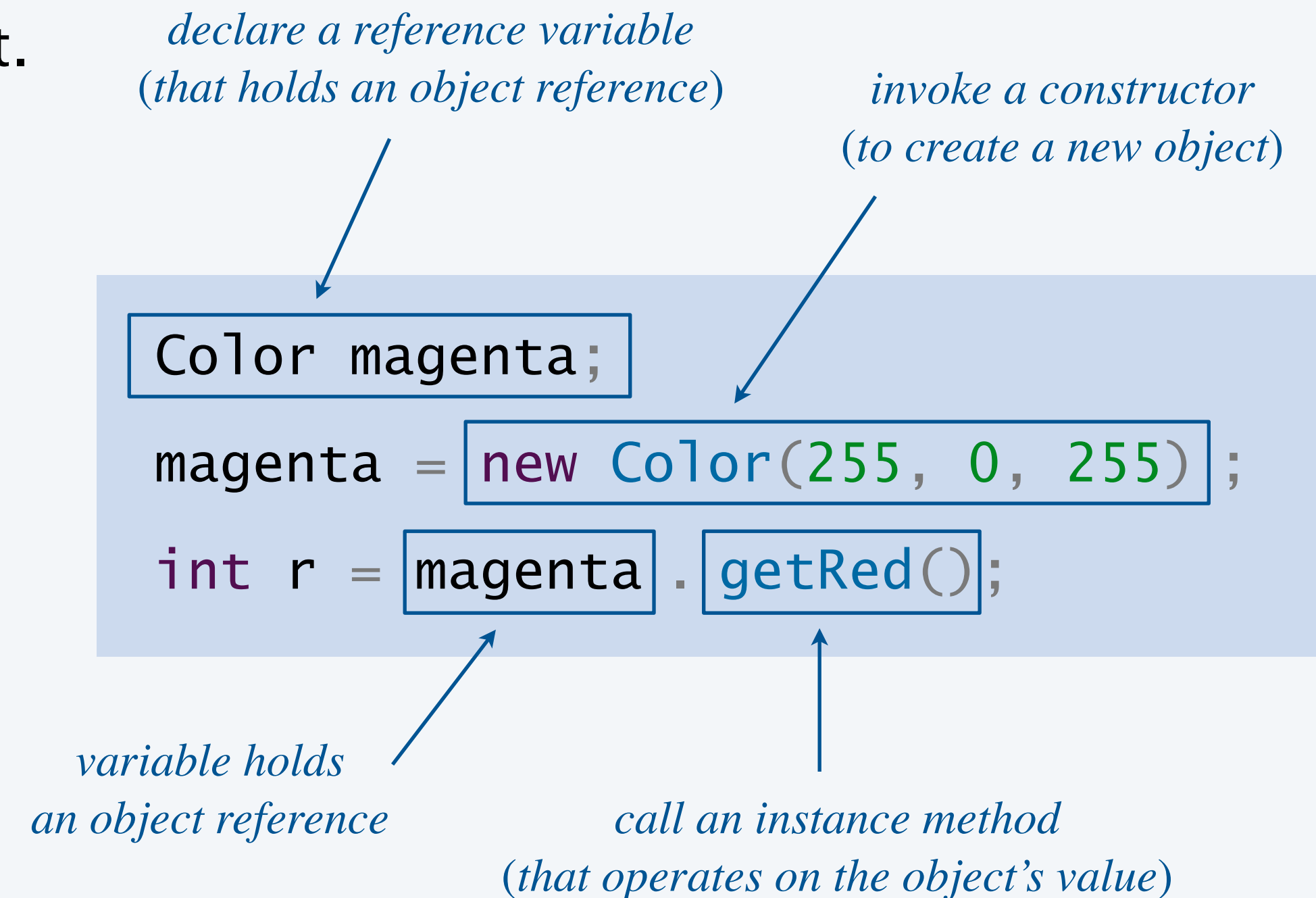
---

## To construct a new object:

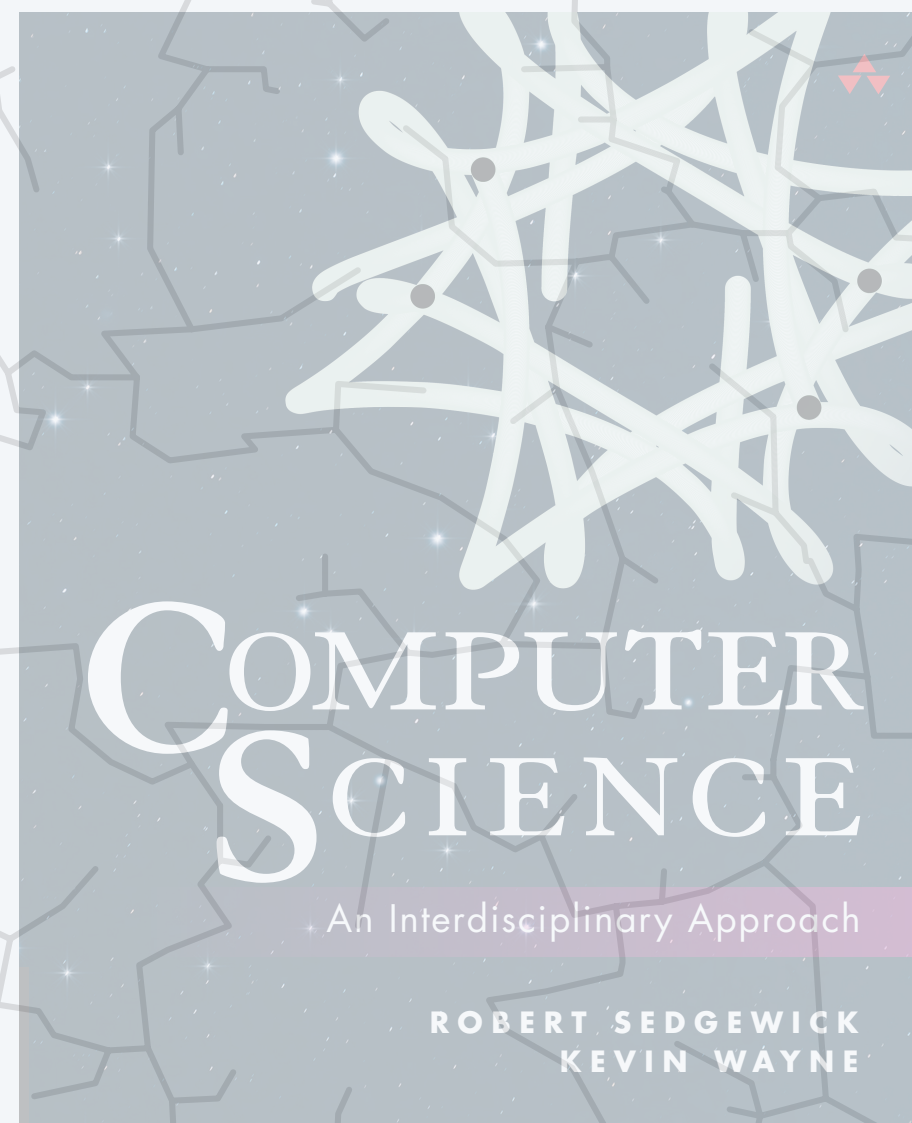
- Use the keyword *new* to invoke a constructor.
- Use **data type name** to specify type of object to construct.
- Include any **arguments**.

## To apply an operation to a given object:

- Use an **object reference** to specify which object.
- Use the **dot operator**.
- Use a **method name** to specify which operation.
- Include any **arguments**.







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

## 3.1 USING DATA TYPES

---

- ▶ *overview*
- ▶ *string processing*
- ▶ *color*
- ▶ *image processing*

# The *String* and *char* data types

A **character** is an individual letter, number, or symbol.

A **string** is a sequence of characters.

Important fundamental abstraction.

- Programming systems (e.g., Java code).
- Communication systems (e.g., text messages).
- Genomic sequences.
- ...

0	1	2	3	4	5	6	7	8	9	10	11	12
T	A	G	A	T	G	T	G	C	T	A	G	C

a DNA string

type	set of values	example values	operations
<code>char</code>	<i>characters</i>	'A' 'B' 'C' '6' '!' 'ă'	compare
<code>String</code>	<i>sequences of characters</i>	"Hello, World" "Nǐ hǎo"	length, concatenate, compare, extract substring, search, ...

**Note.** Java provides special syntax for creating *String* objects. ← *string literals and + operator (instead of new)*



**String data type.** Java includes a *String* data type for manipulating strings.

<code>public class String</code>	<b>description</b>
<code>String(char[] values)</code>	<i>create new string from character array</i>
<code>int length()</code>	<i>length of string</i>
<code>char charAt(int i)</code>	<i>character at index i</i>
<code>boolean startsWith(String pre)</code>	<i>does string start with pre ?</i>
<code>boolean endsWith(String post)</code>	<i>does string end with post ?</i>
<code>boolean equals(Object obj)</code>	<i>do two strings correspond to same sequence of characters?</i>
<code>int indexOf(String t)</code>	<i>index of first occurrence of t</i>
<code>int lastIndexOf(String t)</code>	<i>index of last occurrence of t</i>
<code>String concat(String t)</code>	<i>concatenation of this string and t</i>
<code>String substring(int i, int j)</code>	<i>substring containing characters at indices i through j-1</i>
<code>String replace(char from, char to)</code>	<i>replace all occurrence of character from with to</i>
<code>⋮</code>	<code>⋮</code>

*typically use + operator instead* (points to `concat`)

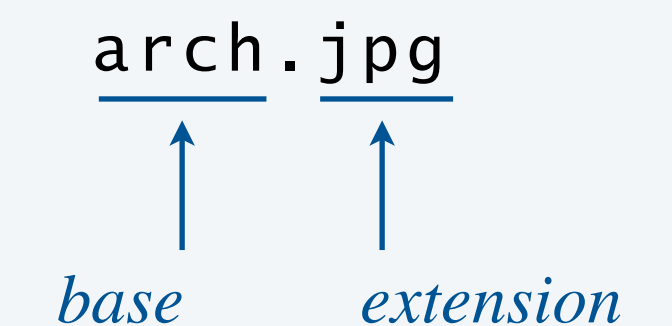
*creates and returns a new String (does not mutate existing string)* (points to `substring`)

# Examples of using the *String* data type

computation	Java code
<i>is the string a palindrome? (string equal to its reverse)</i>	<pre>public static boolean isPalindrome(String s) {     int n = s.length();     for (int i = 0; i &lt; n/2; i++)         if (s.charAt(i) != s.charAt(n-1-i))             return false;     return true; }</pre>
<i>translate from DNA to mRNA (replace letter 'T' with 'U')</i>	<pre>public static String translate(String dna) {     String rna = dna.replace('T', 'U');     return rna; }</pre>
<i>extract base and extension from filename</i>	<pre>String filename = args[0]; int dot = filename.lastIndexOf("."); String base = filename.substring(0, dot); String extension = filename.substring(dot + 1, s.length());</pre>

yes	no
"noon"	"126"
"ACTATCA"	"ACTA"

DNA	mRNA
"ACTG"	"ACUG"
"TTTAG"	"UUUAG"





Which is the the result of executing the following code fragment?

- A. I\*E
- B. I\*ER
- C. TI\*ER
- D. TIGER
- E. Run-time exception

```
String s = "TIGER";  
s.substring(1, 4);  
s = s.replace('G', '*');  
StdOut.println(s);
```

# Identifying a potential gene

Pre-genomics era. Sequence a human genome.

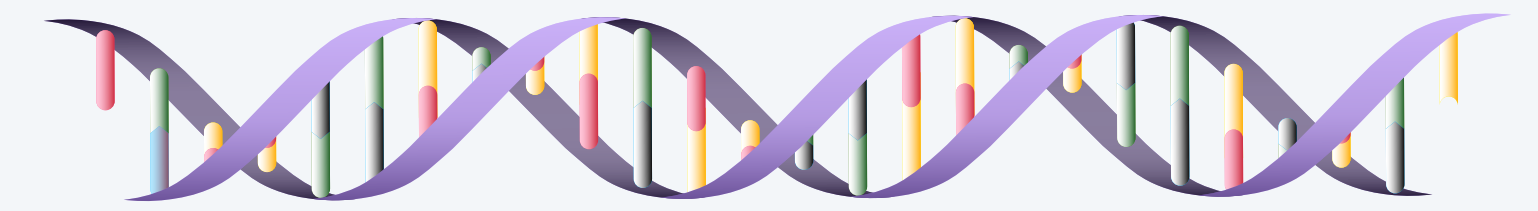
Pre-genomics era. Analyze the data and understand structure.

Genomics. Represent genome as a string over A C T G alphabet.

Gene. A substring of genome that represents a functional unit.

- Made up of **codons** (three A C T G nucleotides).
- Begins with **start codon** ( A T G ).
- Ends with a **stop codon** ( T A G , T A A , or T G A ).
- No intervening stop codons.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	T	G	C	A	T	A	G	C	G	C	A	T	A	G
— start —			— no intervening stop codons —									— stop —		



DNA sequence	potential gene?
ATGCATAGCGCATAG	yes
ATGCGCTGCGTCTGTACTAG	no
ATGCCGTGACGTCTGTACTAG	no

↑  
intervening stop codon

↖ 20 nucleotides (not a multiple of 3)

# Identifying a potential gene

**Goal.** Determine whether a given DNA string is a potential gene.

```
public static boolean isPotentialGene(String dna) {
```

```
    if (dna.length() % 3 != 0) return false;
```

*← length is not a multiple of 3*

```
    if (!dna.startsWith("ATG")) return false;
```

*← does not begin with a start codon*

```
    for (int i = 3; i < dna.length() - 3; i += 3) {
```

```
        String codon = dna.substring(i, i+3);
```

```
        if (codon.equals("TAA")) return false;
```

```
        if (codon.equals("TAG")) return false;
```

*← intervening stop codons*

```
        if (codon.equals("TGA")) return false;
```

```
    }
```

```
    if (dna.endsWith("TAA")) return true;
```

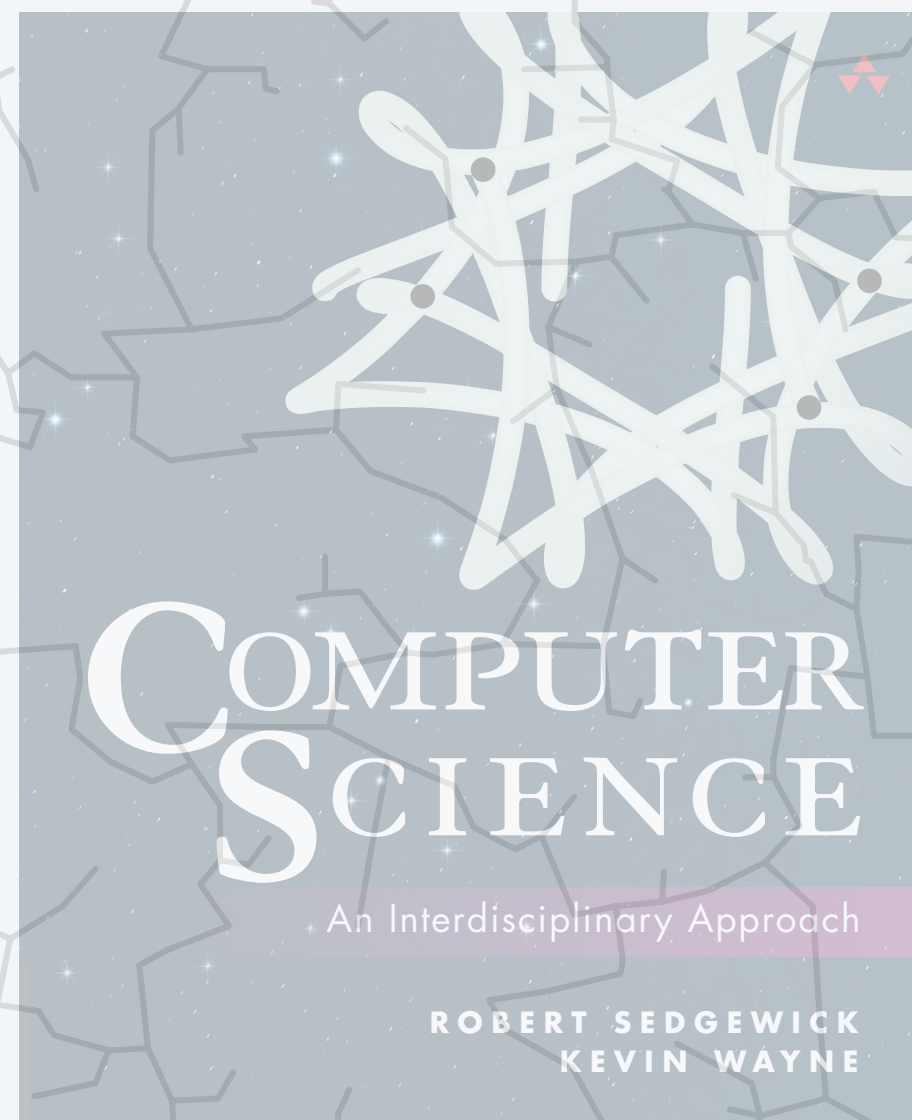
```
    if (dna.endsWith("TAG")) return true;
```

```
    if (dna.endsWith("TGA")) return true;
```

*← ends with a stop codon*

```
    return false;
```

```
}
```



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

## 3.1 USING DATA TYPES

---

- ▶ *overview*
- ▶ *string processing*
- ▶ *color*
- ▶ *image processing*



# Review: RGB color model



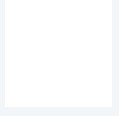

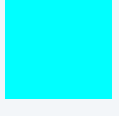

---

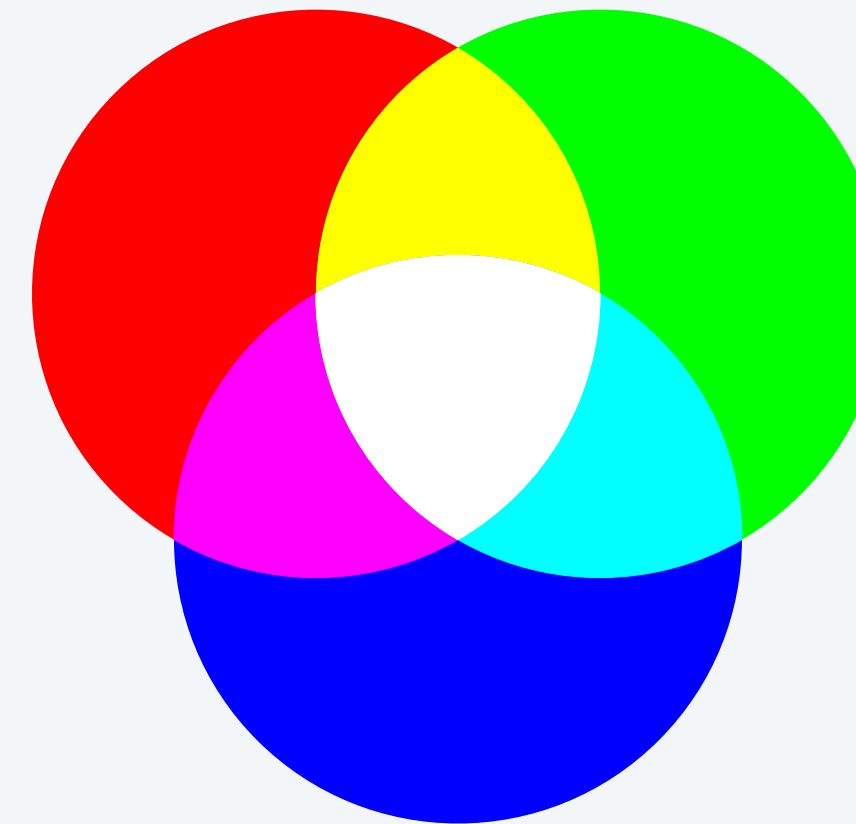
**Color** is a sensation in the eye from electromagnetic radiation.

**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	





**Color data type.** Java includes a *Color* data type for manipulating colors.

<code>public class Color</code>	<b>description</b>
<code>Color(int r, int g, int b)</code>	<i>create a new color with given RGB components</i>
<code>int getRed()</code>	<i>red intensity</i>
<code>int getGreen()</code>	<i>green intensity</i>
<code>int getBlue()</code>	<i>blue intensity</i>
<code>Color brighter()</code>	<i>brighter version of this color</i>
<code>Color darker()</code>	<i>darker version of this color</i>
<code>boolean equals(Object other)</code>	<i>do the two color objects correspond to same RGB values?</i>
<code>String toString()</code>	<i>string representation of this color</i>
<code>⋮</code>	<code>⋮</code>

**Java library.** It's located in *java.awt.Color*, so you need an *import* statement to use.

# Albers squares

---

Josef Albers. A 20<sup>th</sup> century artist who revolutionized the way people think about color.



Homage to the Square (series)

Josef Albers

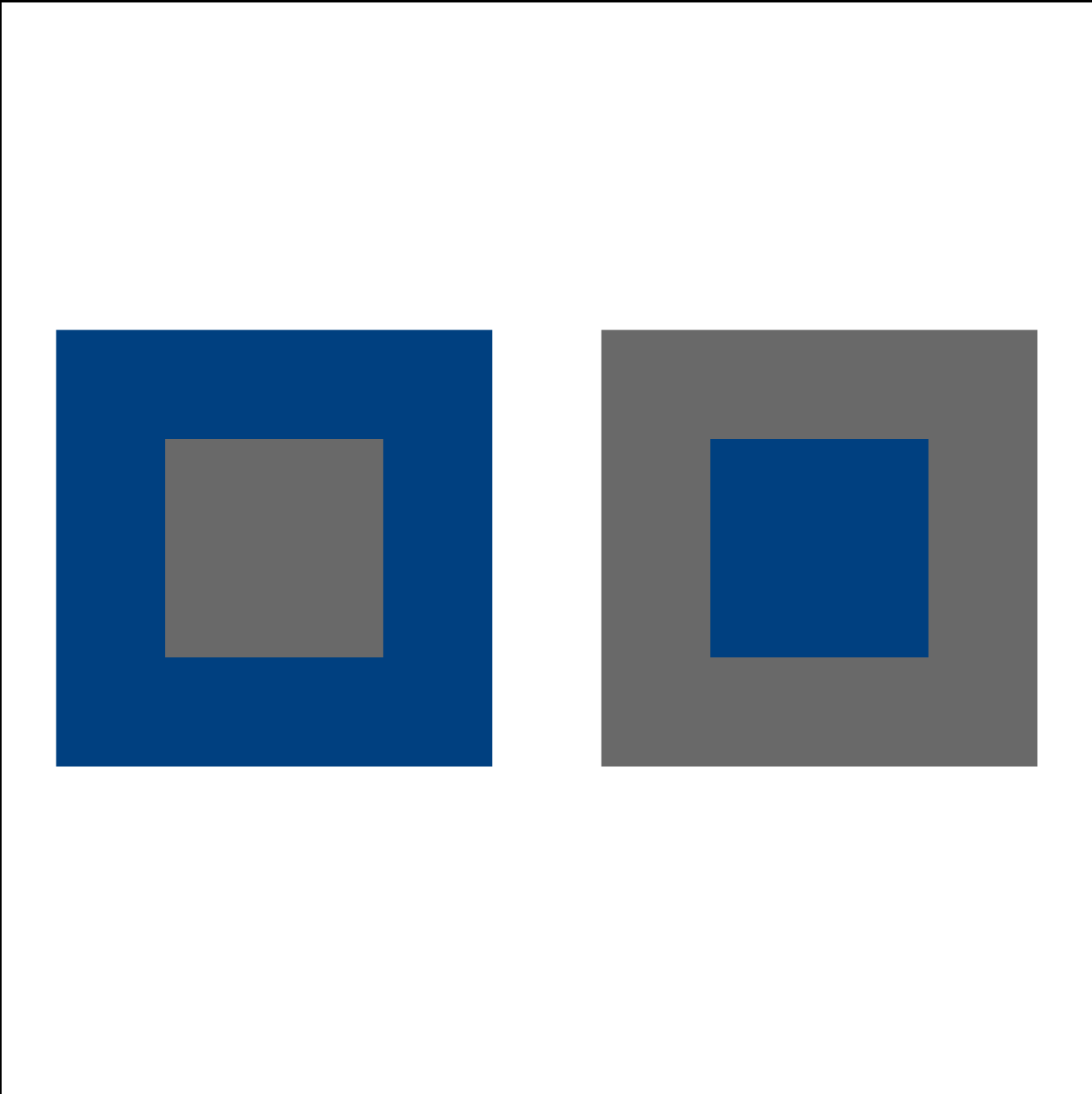
# Albers squares

---

**Goal.** Write a Java program to generate Albers squares.

```
~/cos126/oop1> java-introcs AlbersSquares 0 64 128 105 105 105
```

                          
          ↑        ↑  
 $(r_1, g_1, b_1)$     $(r_2, g_2, b_2)$



The image shows two Albers squares side-by-side. The left square has a blue outer ring and a gray inner square. The right square has a gray outer ring and a blue inner square. The squares are displayed within a white rectangular area on a black background.

# Albers squares implementation

```
import java.awt.Color;

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

```
        int r1 = Integer.parseInt(args[0]);
        int g1 = Integer.parseInt(args[1]);
        int b1 = Integer.parseInt(args[2]);
        Color c1 = new Color(r1, g1, b1);
```

*create first Color object*

```
        int r2 = Integer.parseInt(args[3]);
        int g2 = Integer.parseInt(args[4]);
        int b2 = Integer.parseInt(args[5]);
        Color c2 = new Color(r2, g2, b2);
```

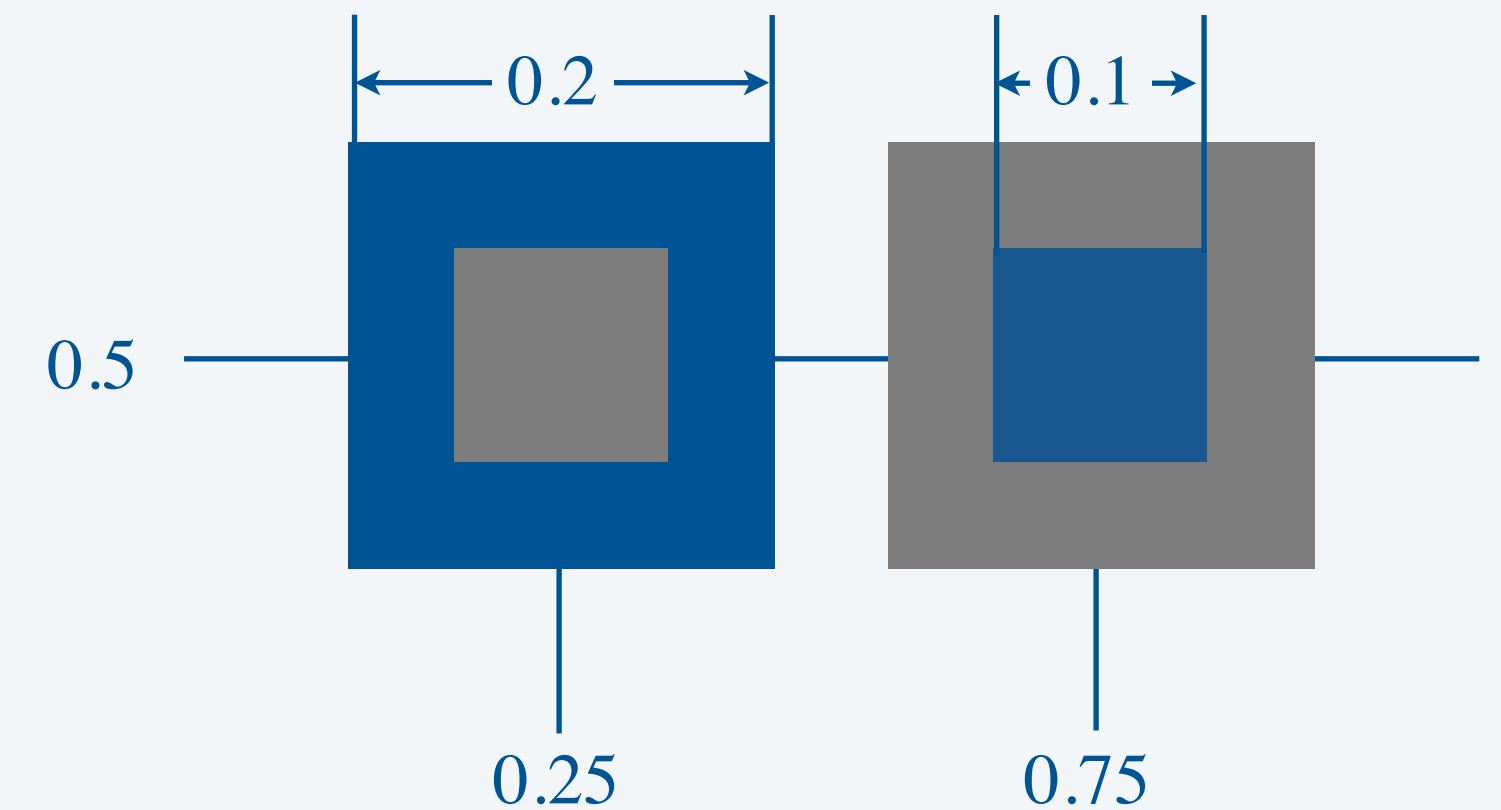
*create second Color object*

```
        StdDraw.setPenColor(c1);
        StdDraw.filledSquare(0.25, 0.5, 0.2);
        StdDraw.setPenColor(c2);
        StdDraw.filledSquare(0.25, 0.5, 0.1);
```

*pass Color object to StdDraw.setPenColor()*

*draw first pair of squares*

```
        StdDraw.setPenColor(c2);
        StdDraw.filledSquare(0.75, 0.5, 0.2);
        StdDraw.setPenColor(c1);
        StdDraw.filledSquare(0.75, 0.5, 0.1);
    }
}
```



# Monochrome luminance




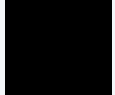


**Def.** The **luminance** of a color quantifies its effective brightness. ← on a scale of 0 (black) to 255 (white)

**Standard formula.**  $Y = 0.299R + 0.587G + 0.114B$ . ← pure green appears lighter than pure blue (so give higher weight)

```
import java.awt.Color;
public class Luminance {
    public static double intensity(Color color) {
        int r = color.getRed();
        int g = color.getGreen();
        int b = color.getBlue();
        return 0.299*r + 0.587*g + 0.114*b;
    }
    public static void main(String[] args) {
        int r = Integer.parseInt(args[0]);
        int g = Integer.parseInt(args[1]);
        int b = Integer.parseInt(args[2]);
        Color color = new Color(r, g, b);
        StdOut.println(intensity(color));
    }
}
```

function takes a Color object as an argument

```
~/cos126/oop1> java-introcs Luminance 255 0 0
76.245
~/cos126/oop1> java-introcs Luminance 0 64 128
52.16
```

name	R	G	B	color	lum
<i>red</i>	255	0	0		76.245
<i>green</i>	0	255	0		149.685
<i>blue</i>	0	0	255		29.07
<i>black</i>	0	0	0		0.0
<i>white</i>	255	255	255		255.0
<i>book blue</i>	0	64	128		52.16

# Foreground/background color accessibility

**Goal.** Determine whether text in one color will be readable if background is in another color.

**Application.** Make web content accessible.

**Web standard.** Readable if **contrast ratio**  $\frac{lum_{max} + 0.05}{lum_{min} + 0.05} \geq 4.5$ .

↑  
*WCAG uses relative luminance,  
not monochrome luminance*



**Web Content  
Accessibility Guidelines**

## Luminance.java

```
public static double contrastRatio(Color a, Color b) {  
    double min = Math.min(intensity(a), intensity(b)) / 255.0;  
    double max = Math.max(intensity(a), intensity(b)) / 255.0;  
    return (max + 0.05) / (min + 0.05);  
}
```

```
public static boolean isAccessible(Color a, Color b) {  
    return contrastRatio(a, b) >= 4.5;  
}
```

← *normalized to be  
between 0 and 1*

1.7	1.7
2.1	2.1
3.0	3.0
8.6	8.6
21	21

**contrast ratios  
(between 1 and 21)**

# Grayscale

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






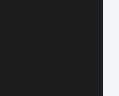
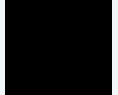





- RGB color is gray when  $R = G = B$ .
- To convert RGB color to grayscale, use **luminance** for  $R$ ,  $G$ , and  $B$ .



Luminance.java

```
public static Color toGray(Color c) {  
    int y = (int) Math.round(intensity(c));  
    Color gray = new Color(y, y, y);  
    return gray;  
}
```

*round to  
nearest int*

name	R	G	B	color	lum	gray
<i>red</i>	255	0	0		76.245	
<i>green</i>	0	255	0		149.685	
<i>blue</i>	0	0	255		29.07	
<i>black</i>	0	0	0		0.0	
<i>white</i>	255	255	255		255.0	
<i>book blue</i>	0	64	128		52.16	





# Object references: memory representation

**Object reference.** Refers to a data-type value; it is not the value.

*can think of an object reference as the memory address of an object*

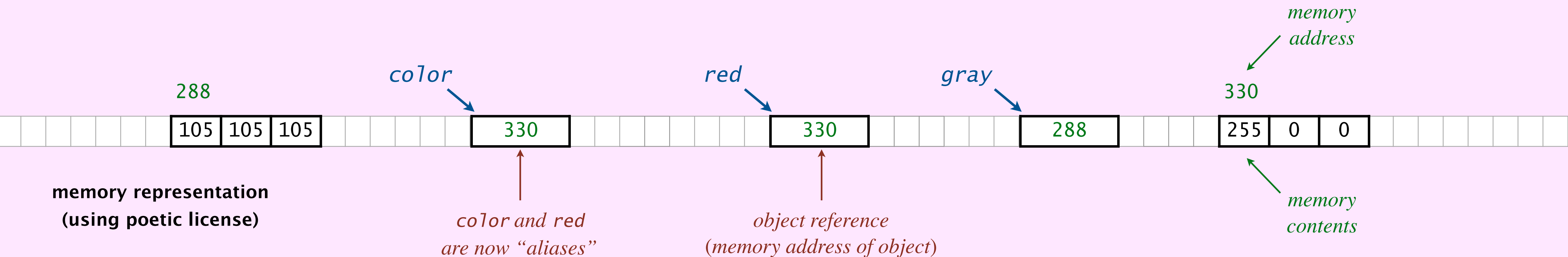
- Can manipulate the value in the object it refers to.
- Can use it to invoke instance methods (with the . operator).
- Can pass it to (or return it from) a method.

```

Color red = new Color(255, 0, 0);
Color gray = new Color(105, 105, 105);
Color color = red;

```

*the reference variables red, gray, and color store object references*



memory representation (using poetic license)

*color and red are now "aliases"*

*object reference (memory address of object)*

*memory contents*

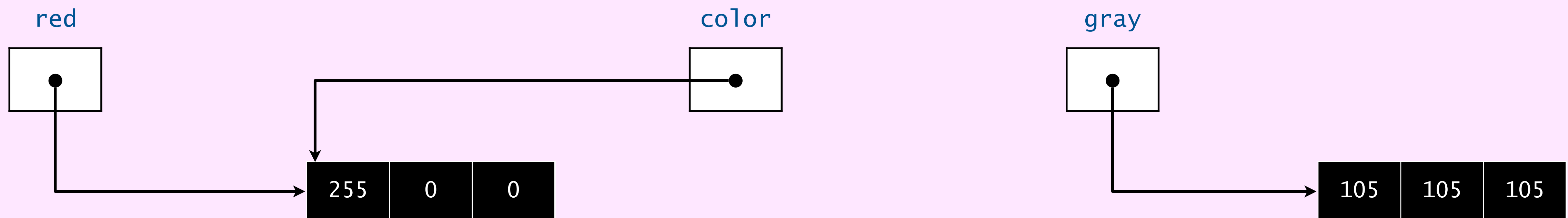


# Object references: box-and-pointer diagrams

## Box-and-pointer diagram.

- Put each object and reference variable in a box.
- Draw an arrow from each reference variable to the object it references.

```
Color red = new Color(255, 0, 0);  
Color gray = new Color(105, 105, 105);  
Color color = red;
```





Assume that the variables `red1`, `red2`, and `red3` are initialized as follows.

Which of the following expressions will evaluate to `false` ?

- A. `red1 == red3`
- B. `red2 == red3`
- C. `red1.equals(red3)`
- D. `red2.equals(red3)`

```
Color red1 = new Color(255, 0, 0);  
Color red2 = new Color(255, 0, 0);  
Color red3 = red1;
```

# References and abstraction

---

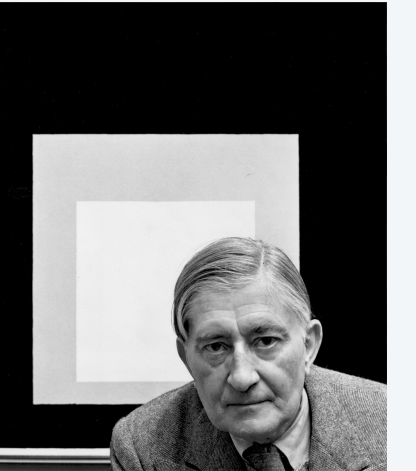
René Magritte. “This is not a pipe.”



← *it is a picture of  
a painting of a pipe*

*“For me, abstraction is real,  
probably more real than nature.”*

*—Josef Albers*



Java. These are not colors.

```
Color red = new Color(255, 0, 0);  
Color gray = new Color(105, 105, 105);
```

← *they are Java  
representations of colors*

OOP. A natural vehicle for studying abstract models of the real world.

# This is not a pipe memes



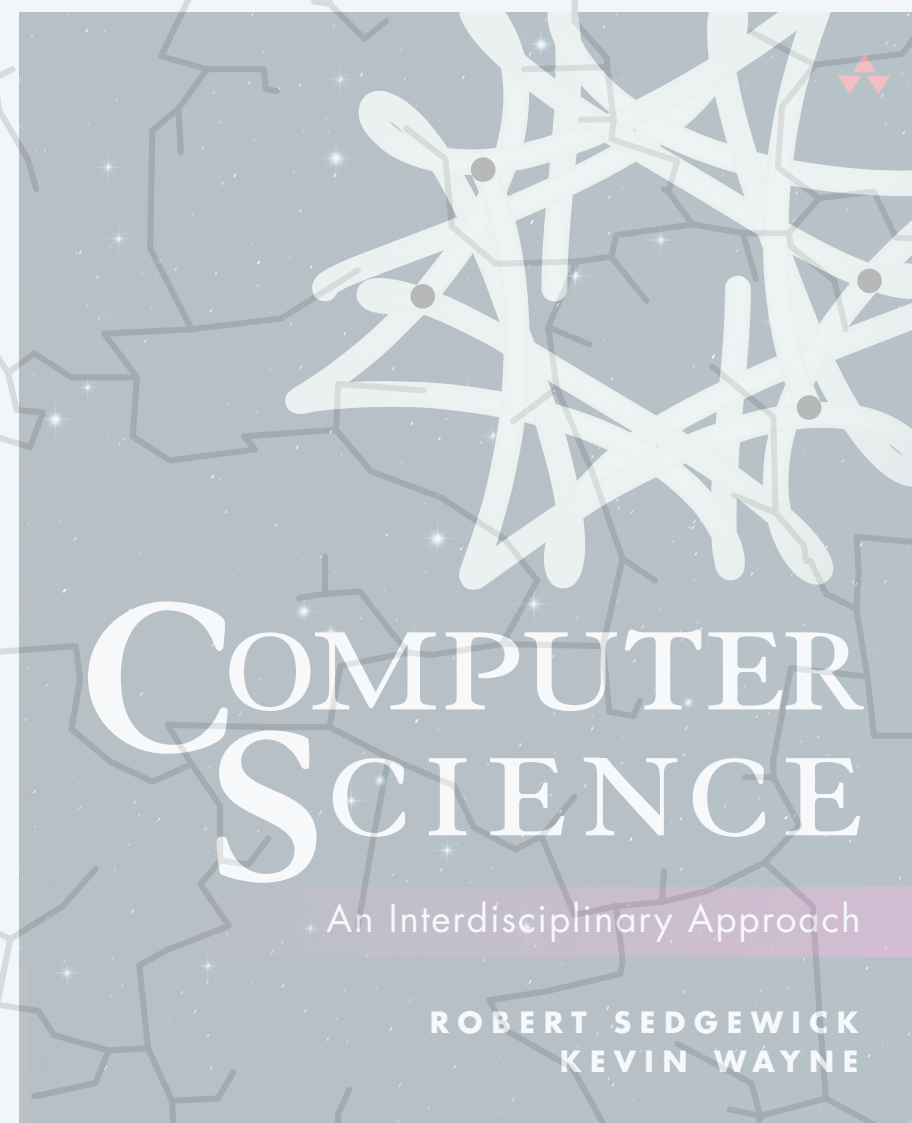
Select all squares with

**pipes**

If there are none, click skip



SKIP



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

## 3.1 USING DATA TYPES

---

- ▶ *overview*
- ▶ *string processing*
- ▶ *color*
- ▶ *image processing*



# Input and output data types

You have used. *StdIn*, *StdOut*, *StdDraw*, and *StdPicture*.

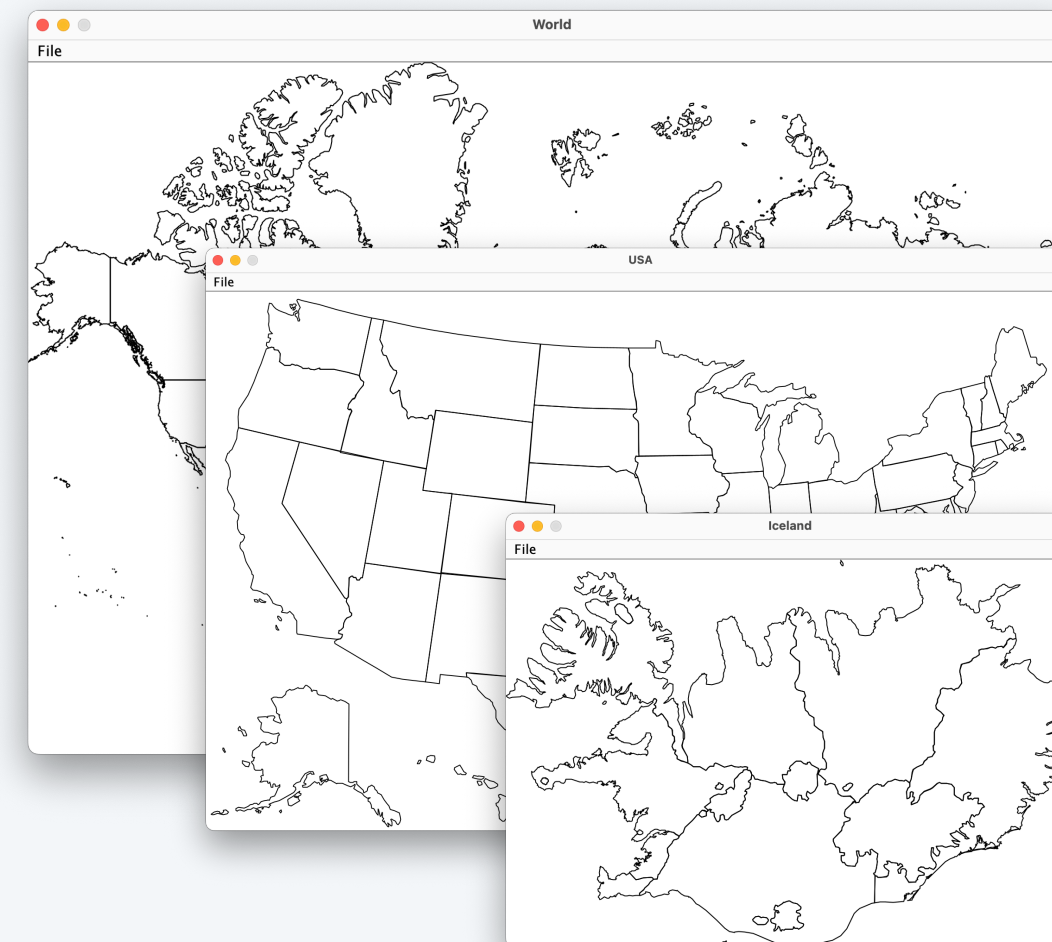
Key limitation. Only one entity per program.

*one input stream, output stream,  
drawing, or picture  
per program execution*

OOP versions. We also provide object-oriented versions.

*available with javac-introcs  
and java-introcs commands*

data type	enables
In	<i>read from more than one input stream</i>
Out	<i>write to more than one output stream</i>
Draw	<i>create more than one drawing</i>
Picture	<i>process more than one image</i>



# Image processing: review

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

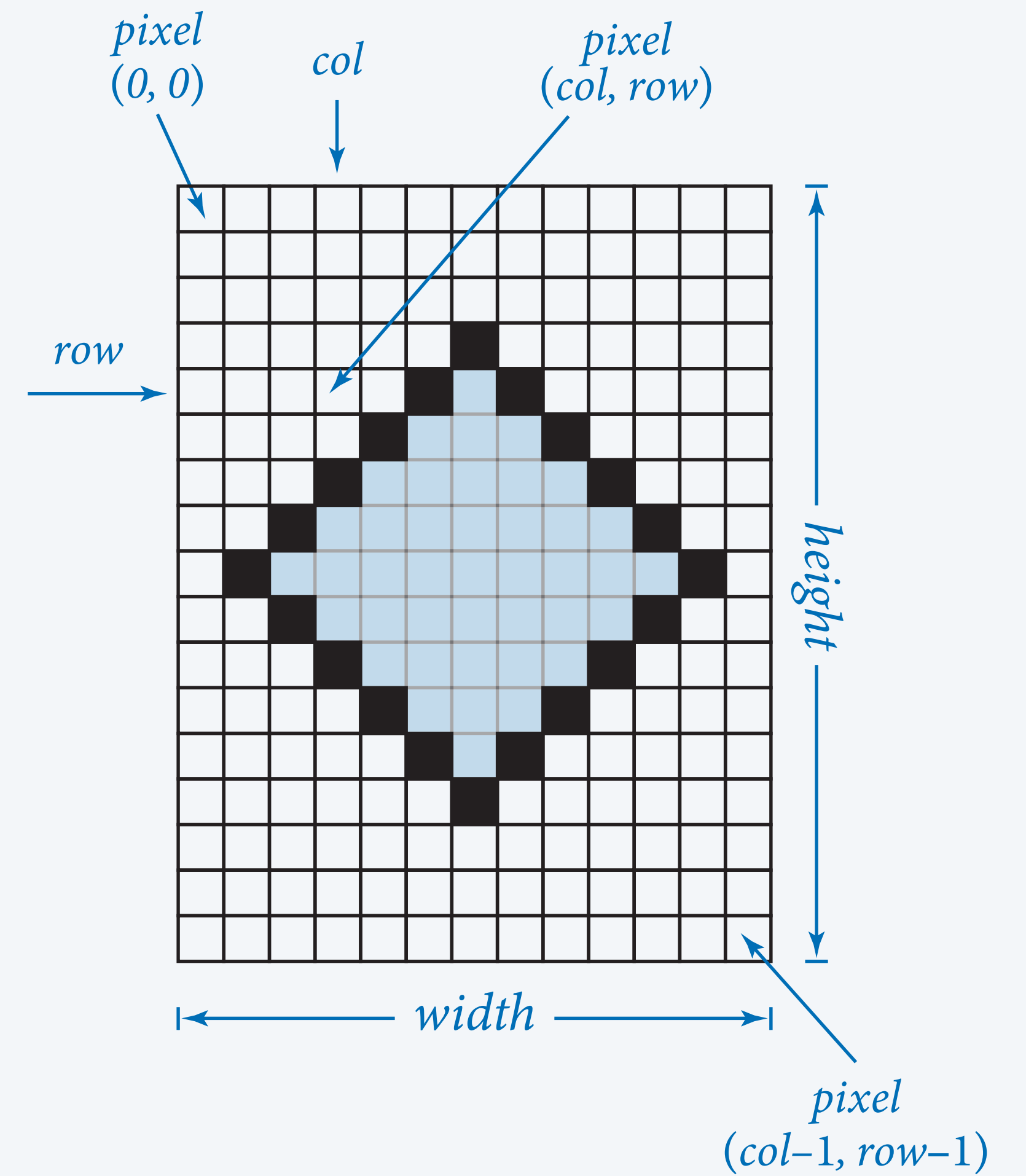
Ex.



mandrill.jpg



arch.jpg





**Picture data type.** Our data type for manipulating digital images.

- Can create many *Picture* objects in same program.
  - Uses *Color* objects as arguments and return values.
- ← *OOP version of StdPicture  
(with a few important differences)*

```
public class Picture
```

**description**

```
    Picture(String filename)
```

*create a picture from an image file*

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

```
    Picture(int width, int height)
```

*create a blank width-by-height picture*

```
    int width()
```

*width of the picture*

```
    int height()
```

*height of the picture*

```
    Color get(int col, int row)
```

*the color of pixel (col, row)*

```
    void set(int col, int row, Color color)
```

*set the color of pixel (col, row) to color*

```
    void show()
```

*display the image in its own window*

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

*save the picture to a file*

# Grayscale filter

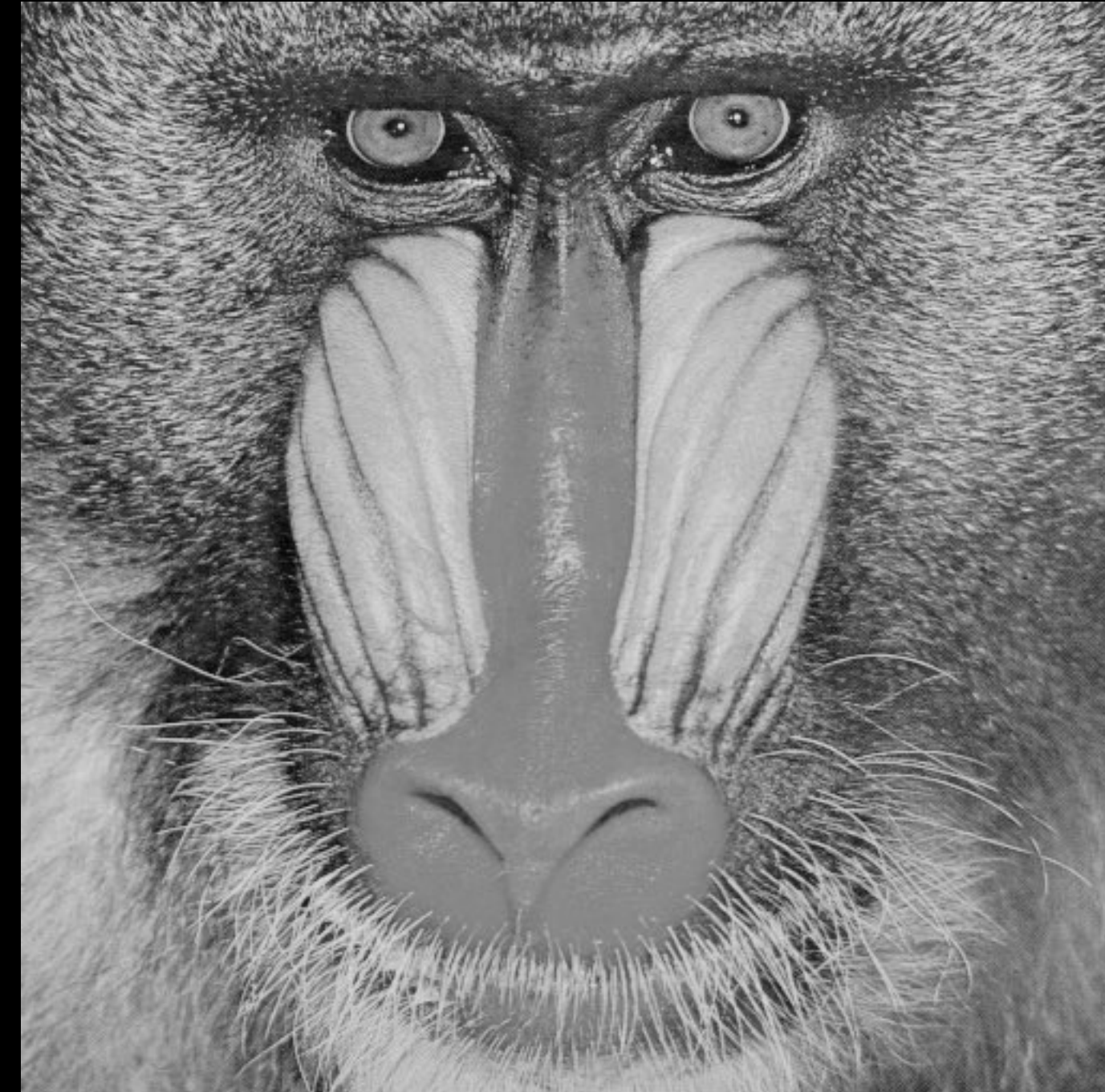
---

**Goal.** Write a Java program to convert an image to grayscale.

```
~cos126/oop1> java-introcs Picture mandrill.jpg
```



```
~cos126/oop1> java-introcs Grayscale mandrill.jpg
```



# Grayscale filter implementation: object-oriented version

---

```
import java.awt.Color;
```

```
public class Grayscale {
```

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

```
        Picture picture = new Picture(args[0]);
```

*← create a new picture  
from image file*

```
        for (int col = 0; col < picture.width(); col++) {
```

```
            for (int row = 0; row < picture.height(); row++) {
```

```
                Color color = picture.get(col, row);
```

```
                Color gray = Luminance.toGray(color);
```

*← change each pixel to grayscale*

```
                picture.set(col, row, gray);
```

```
            }
```

```
        }
```

```
        picture.show();
```

*← display picture  
(in its own window)*

```
    }
```

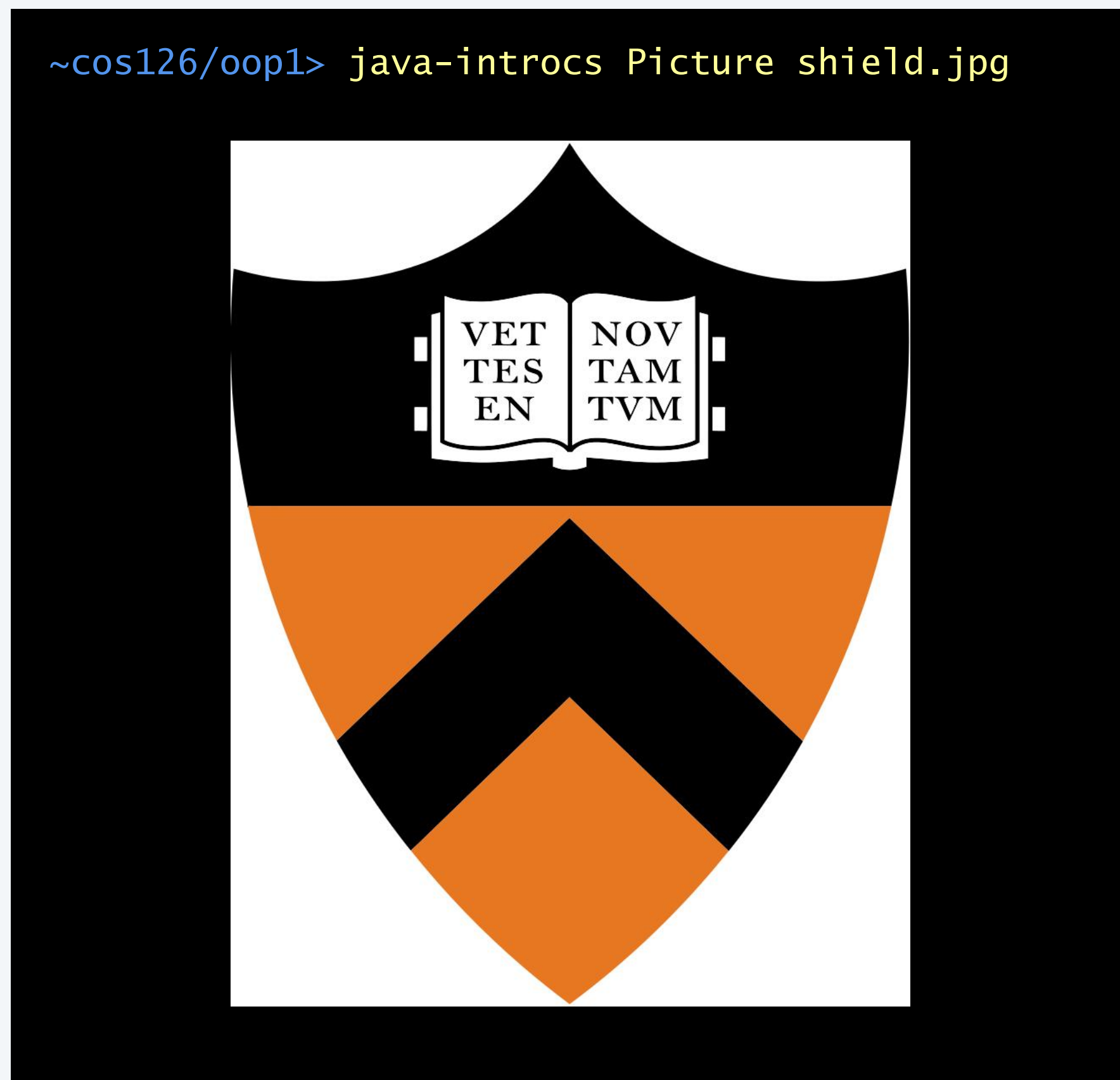
```
}
```

## Rotate an image

---

**Goal.** Write a Java program to create a right-rotated (90° clockwise) version of an image.

**Note.** Need two *Picture* objects (since they are of different dimensions).



# Rotate an image right: demo



**Goal.** Rotate an image right (90° clockwise).

(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)

source image (6-by-4)

# Rotate an image right: demo



**Goal.** Rotate an image right (90° clockwise).

**Algorithm.** Pixel  $(col, row)$  in source image becomes to pixel  $(height - row - 1, col)$  in target 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)

source image (6-by-4)

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

target image (4-by-6)

# Right rotate an image implementation

---

```
import java.awt.Color;

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

        Picture source = new Picture(args[0]);
        int width = source.width();
        int height = source.height();

        Picture target = new Picture(height, width);

        for (int col = 0; col < width; col++) {
            for (int row = 0; row < height; row++) {
                Color color = source.get(col, row);
                target.set(height - row - 1, col, color);
            }
        }

        source.show();
        target.show();
    }
}
```

← *create picture from file  
(and get dimensions)*

← *create a new picture  
(of appropriate dimensions)*

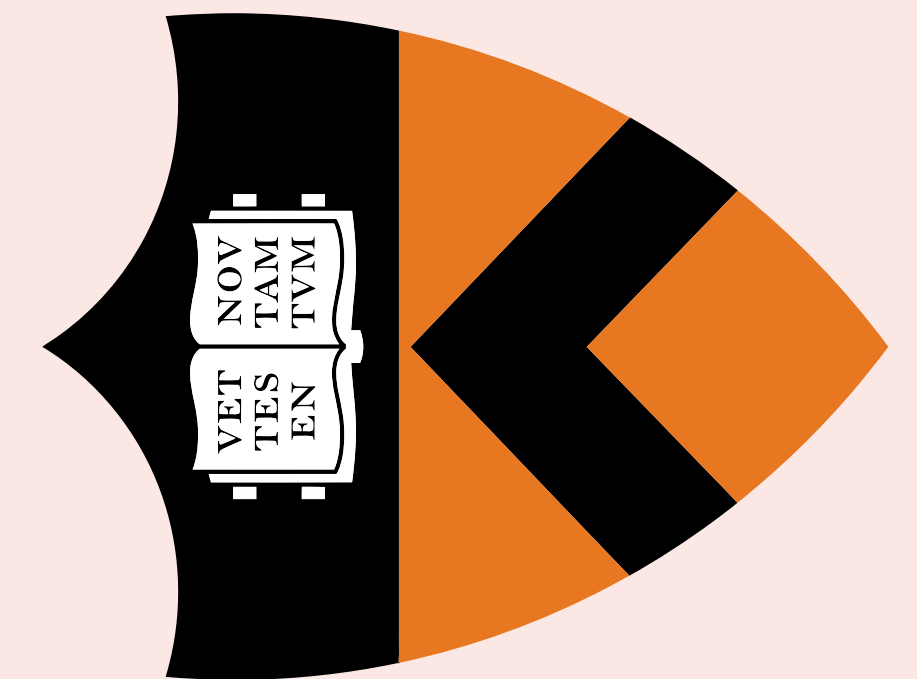
← *process each pixel*

← *display each picture  
(in its own window)*



Fill in the missing code to **left rotate** (90° counterclockwise) an image?

```
for (int col = 0; col < width; col++) {  
    for (int row = 0; row < height; row++) {  
        Color color = source.get(col, row);  
        target.set(████████████████████);  
    }  
}
```

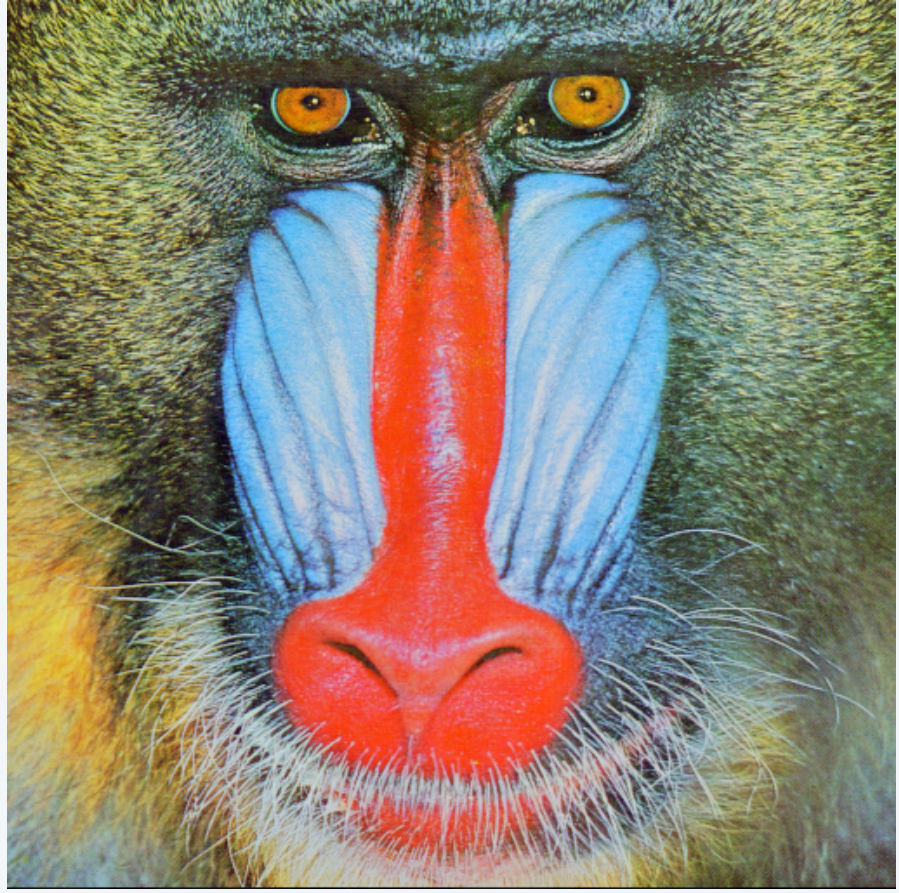


- A. target.set(col, row, color);
- B. target.set(row, col, color);
- C. target.set(height - row - 1, col, color);
- D. target.set(row, width - col - 1, color);



# More image-processing effects

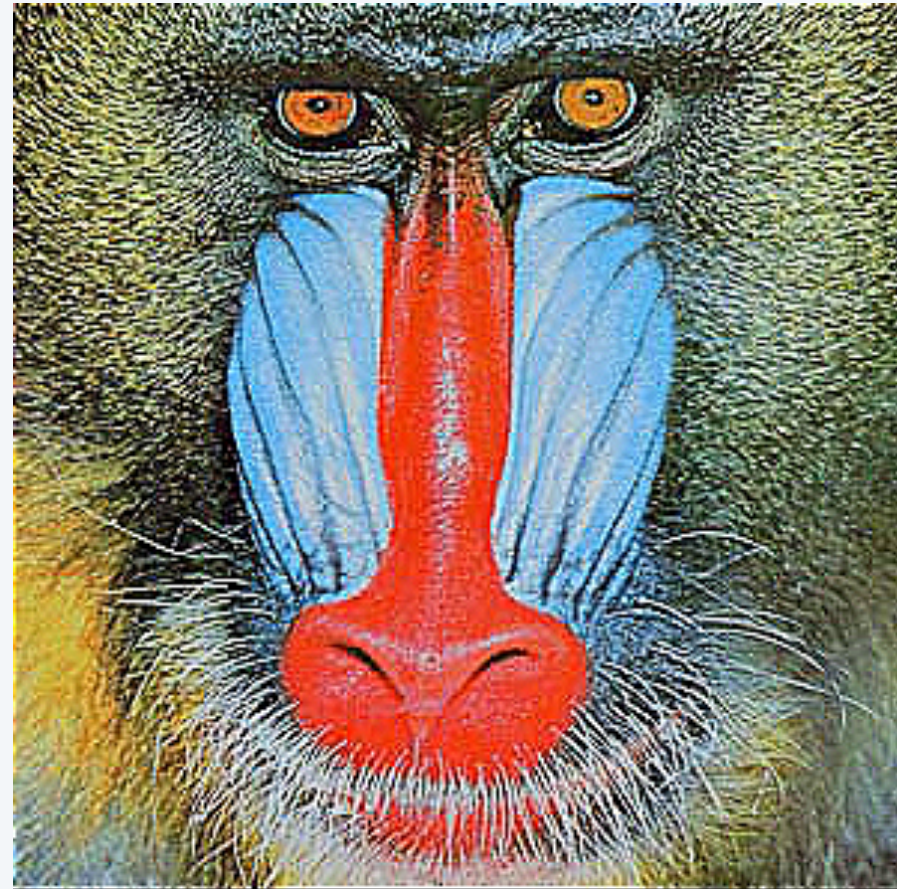
---



original



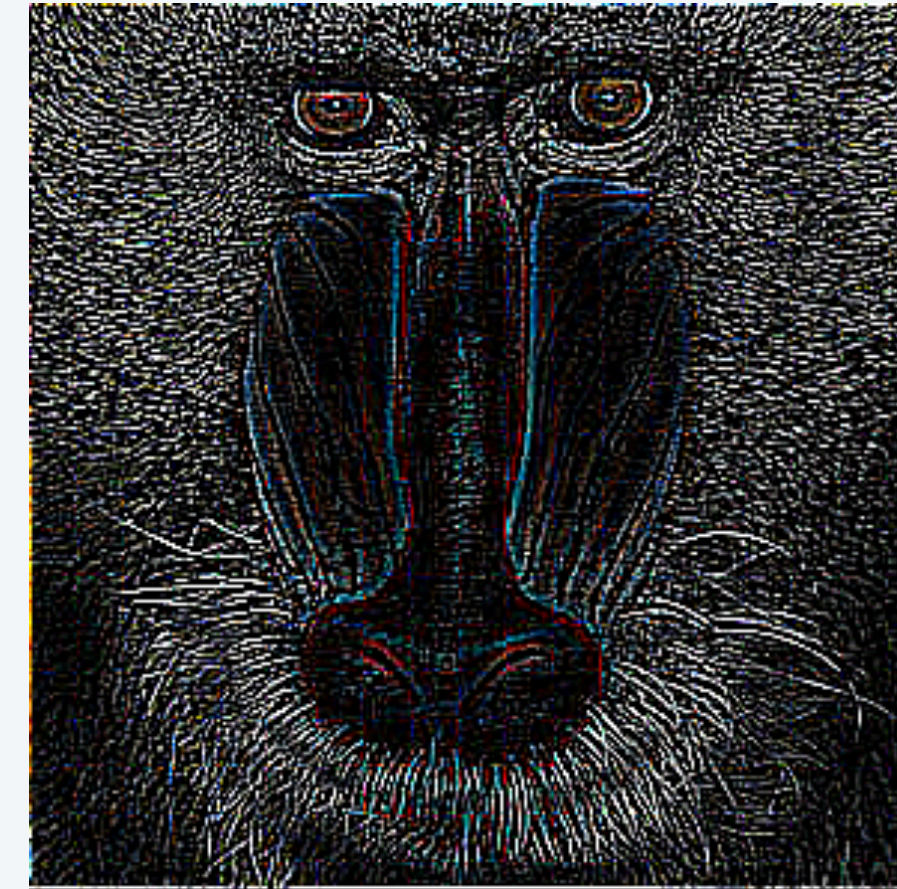
Gaussian blur



sharpen



emboss



Laplacian



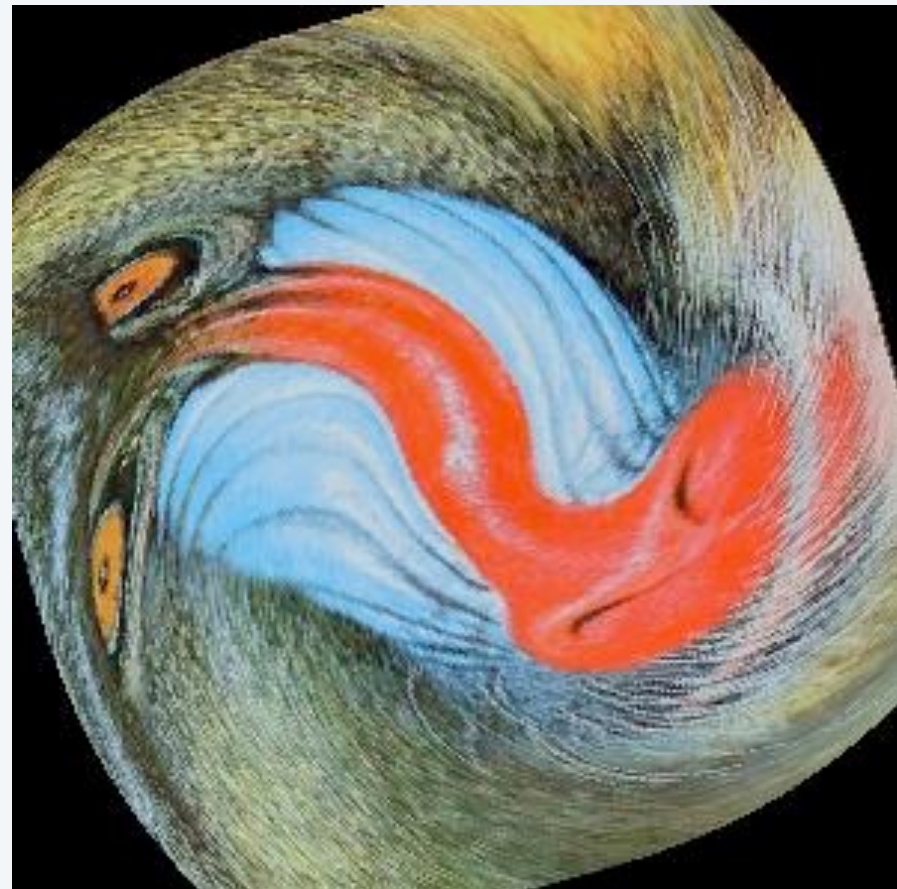
motion blur

# More image-processing effects

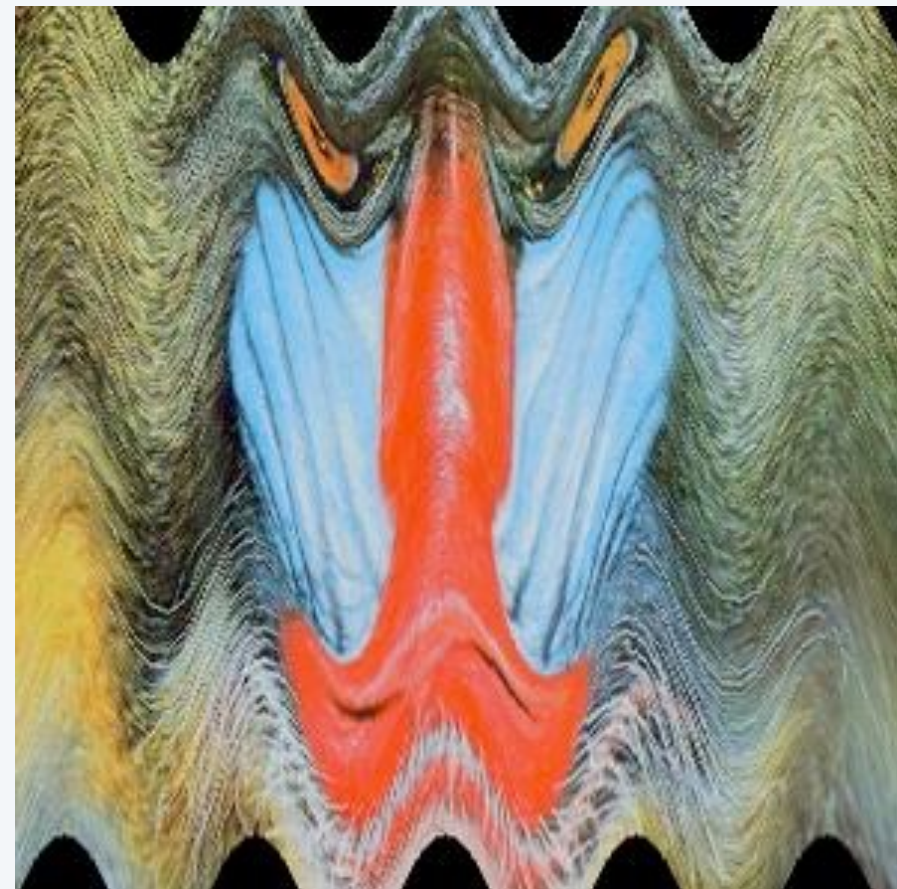
---



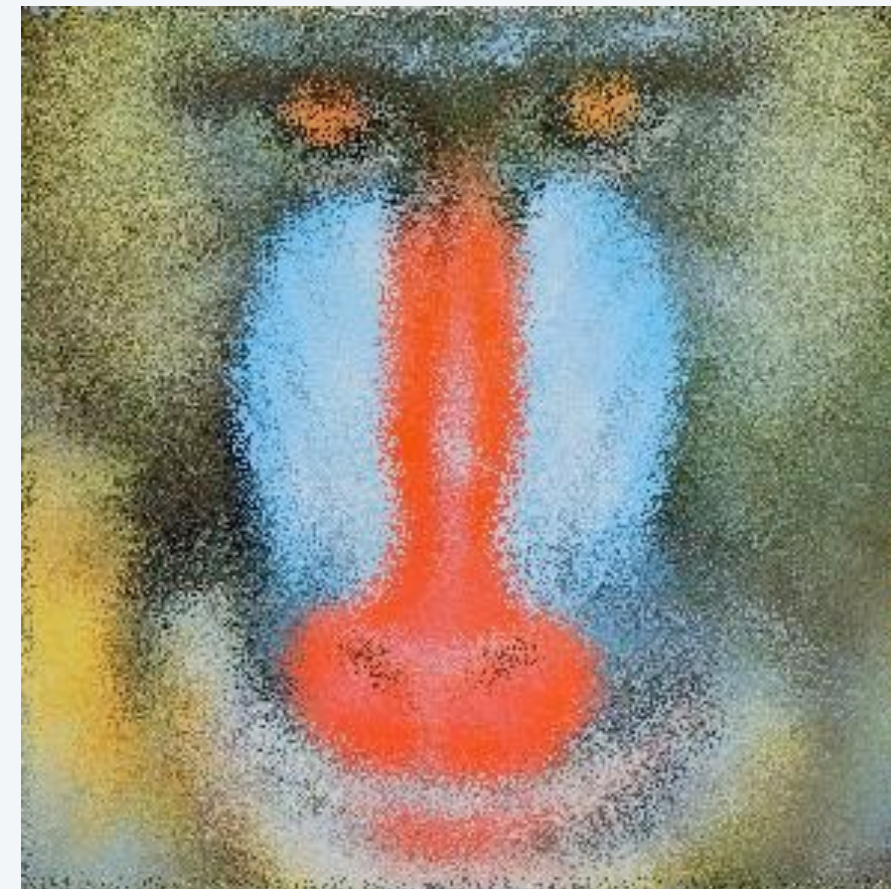
RGB color separation



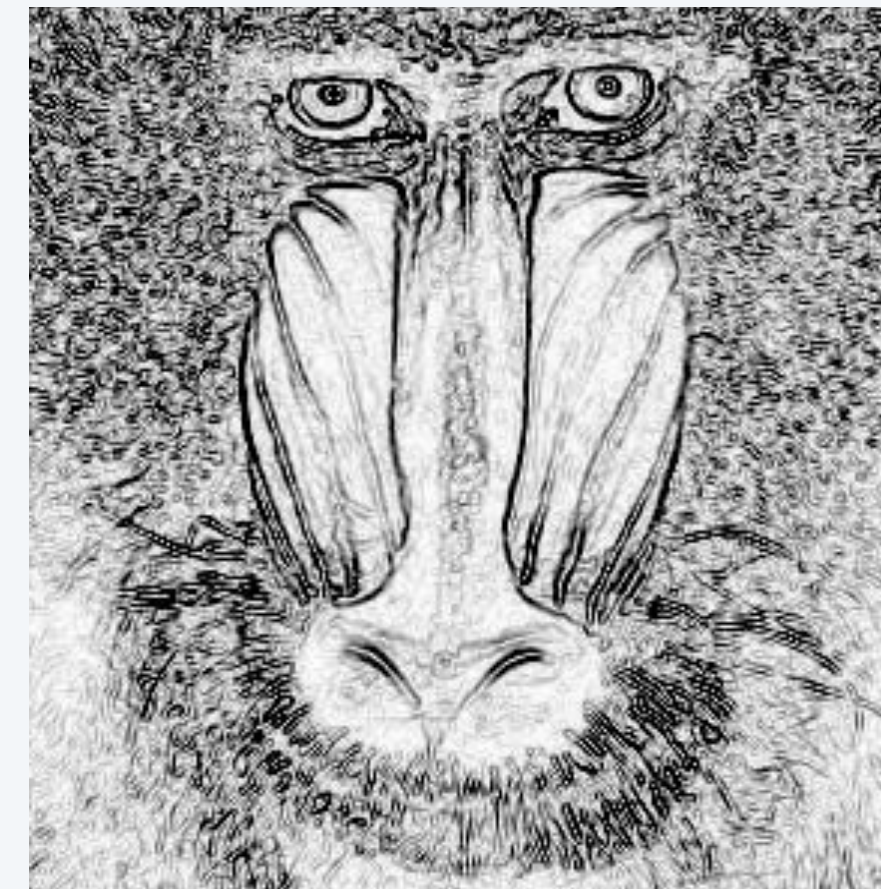
swirl filter



wave filter



glass filter



Sobel edge detection



rescale

# Data types

---

## Object-oriented programming.

- Create your own **data types**.
- Construct and use **objects** in your programs.

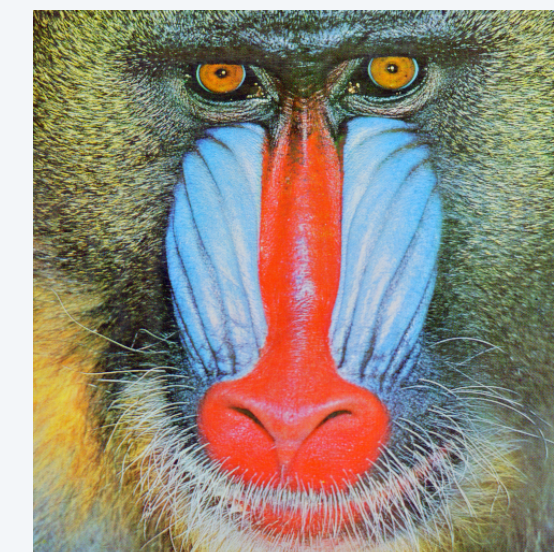
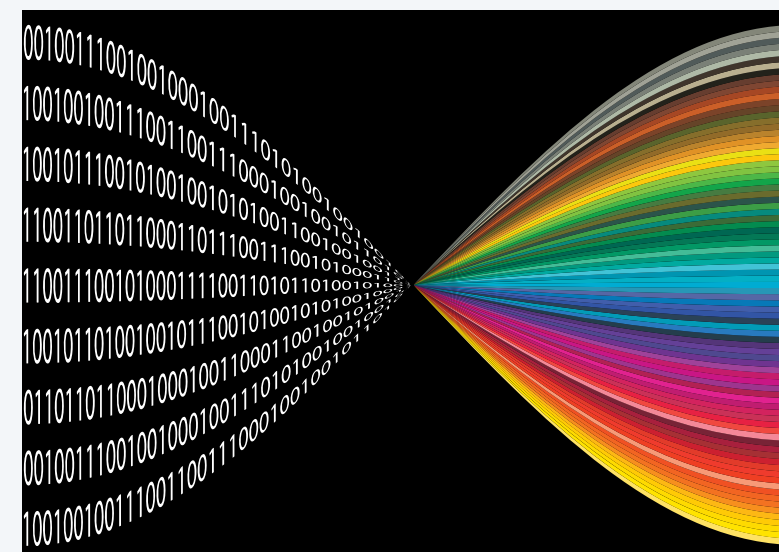
## In Java, programs manipulate object references.

- Almost all data types in Java are reference types. ← `String, Color, Picture, arrays, ...`
- Exceptions: primitive types. ← `int, double, boolean, char, ...`
- OOP purist: languages should have only reference types.

**This lecture.** Use pre-existing data types (for strings, colors, and pictures).

**Next lecture.** Create your own data types.

T	A	G	A	T	G	T	G	C	T	A	G	C
---	---	---	---	---	---	---	---	---	---	---	---	---



# Credits

---

<b>image</b>	<b>source</b>	<b>license</b>
<i>Binary Code of Digital Images</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>CPU Icon</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>OOP Dice</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Molecular Structure of DNA</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>RGB Color Model</i>	<u>Wikimedia</u>	<u>Kopimi</u>
<i>LGBTQ+ Eye</i>	<u>Christian Ibarra Santillan</u>	<u>CC BY 2.0</u>
<i>Josef Albers</i>	<u>Arnold Newman</u>	
<i>Homage to the Square</i>	<u>Josef Albers</u>	

# Credits

---

<b>image</b>	<b>source</b>	<b>license</b>
<i>WCAG 2.0 Compliant</i>	<u>REMOTeI</u>	
<i>The Treachery of Images</i>	<u>René Magritte</u>	
<i>Surrealist Painter and Plumber</i>	<u>Dan Piraro</u>	<u>Educational use</u>
<i>Select All Squares with Pipes</i>	<u>Noah Veltman</u>	
<i>Image Processing Icon</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Mandrill</i>	<u>SIPI Image Database</u>	
<i>Johnson Arch</i>	<u>Danielle Alio Capparella</u>	by photographer
<i>Princeton Shield</i>	<u>Wikimedia</u>	<u>public domain</u>