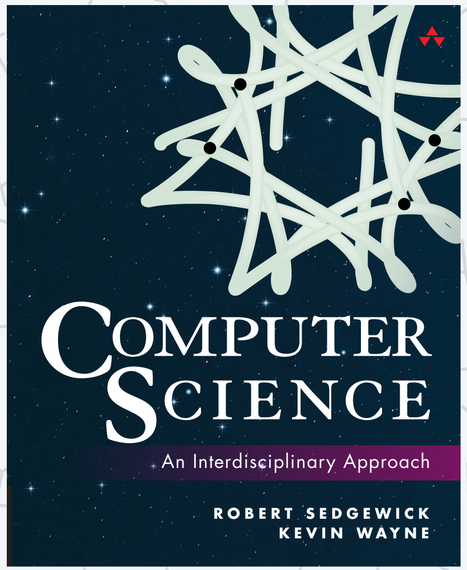
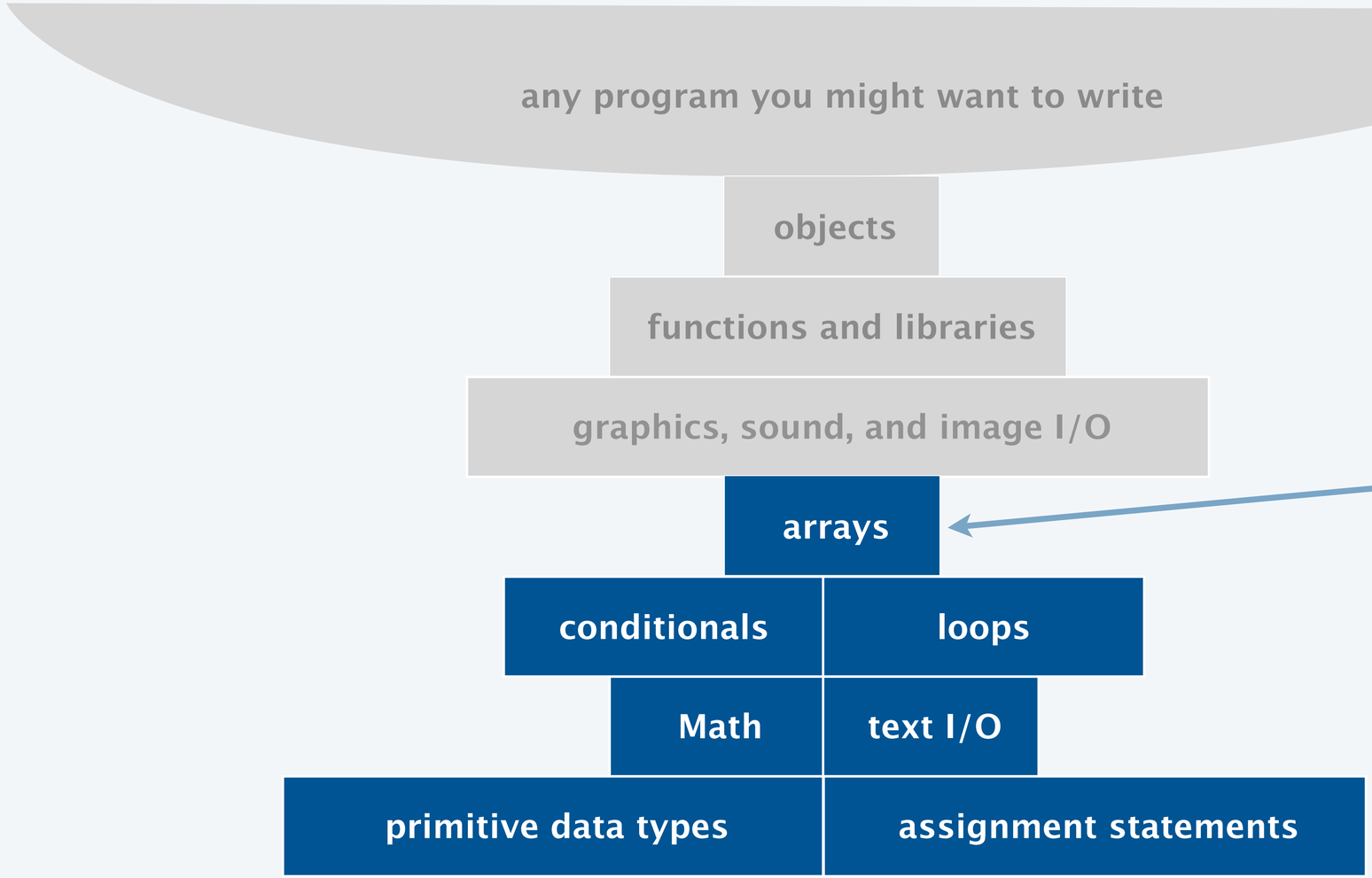



https://introcs.cs.princeton.edu

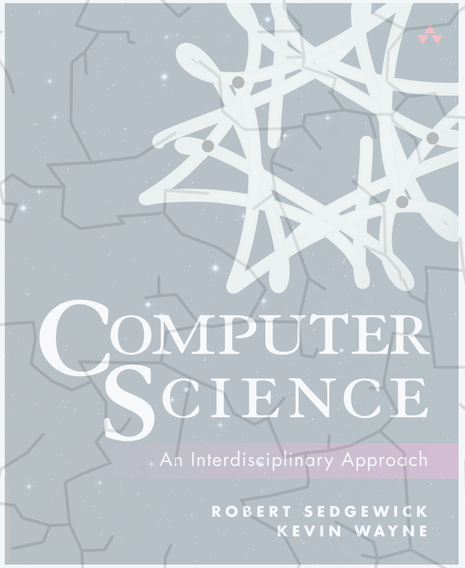
# 1.4 Arrays

- *basic concepts*
- *shuffling*
- *digital audio*
- *memory representation*
- *two-dimensional arrays*

# Basic building blocks for programming




*store and process huge amounts of data*




https://introcs.cs.princeton.edu

# 1.4 ARRAYS

- *basic concepts*
- *shuffling*
- *digital audio*
- *memory representation*
- *two-dimensional arrays*

# Your first data structure

An array is an *indexed sequence* of values of the same type.

Examples.

- 8 notes in a musical scale.
- 52 playing cards in a deck.
- 300 students in a COS class.
- 10 million audio samples in a song.
- 4 billion nucleotides in a DNA strand.
- 100 billion Google queries in a month.
- 1 trillion parameters in a large language model.
- …

| index | value |
|---|---|
| 0 | 2♥ |
| 1 | 6♠ |
| 2 | A♦ |
| 3 | A♥ |
| ⋮ | ⋮ |
| 49 | 3♣ |
| 50 | K♣ |
| 51 | 4♠ |

Main purpose. Facilitate storage and manipulation of data.

# Processing many values of the same type

**10 values, without an array**

```
double a0 = 0.0;
double a1 = 0.0;
double a2 = 0.0;
double a3 = 0.0;
double a4 = 0.0;
double a5 = 0.0;
double a6 = 0.0;
double a7 = 0.0;
double a8 = 0.0;
double a9 = 0.0;
...
a4 = 3.0;
...
a8 = 8.0;
...
double x = a4 + a8;
```

*tedious and error-prone code*

**10 values, with an array**

```
double[] = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```

*an easy alternative*

**1 million values, with an array**

```
double[] a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```

*scales to handle huge amounts of data*

# Arrays in Java

Create an array. Specify its type and length.

Access an array element. Use name of array, square brackets, and index.

| operation | typical code |
|---|---|
| *declare an array* | `double[] a;` |
| *create an array of length n* | `a = new double[n];` |
| *declare, create, and initialize an array* | `double[] b = new double[n];` |
| *array initializer* | `double[] c = { 0.3, 0.6, 0.1 };` |
| *access an array element by index* | `a[i] = b[i-1] + c[i+1];` |
| *length of array* | `a.length` |

← *all elements initialized to default value (zero for numeric types, false for* `boolean`*)*

← *index can be any expression of type* `int`

# Examples of programming with arrays

| problem | code | |
|---|---|---|
| *print array elements, one per line* | `for (int i = 0; i < a.length; i++)`<br>`  System.out.println(a[i]);` | ← *array indices go from* `0` *to* `a.length - 1` |
| *sum of array elements* | `double sum = 0.0;`<br>`for (int i = 0; i < a.length; i++)`<br>`  sum += a[i];` | ← *array elements are variables (can be used in expressions)* |
| *create a new array containing n random numbers* | `double[] a = new double[n];`<br>`for (int i = 0; i < n; i++)`<br>`  a[i] = Math.random();` | ← *array elements are variables (can be used as LHS of assignment statement)* |
| *command-line arguments* | `int time = Integer.parseInt(args[0]);`<br>`String voice = args[1];` | ← `args[]` *in* `main()` *is a* `String` *array* |
| *months in the year* | `String[] months = {`<br>`  "Jan", "Feb", "Mar", "Apr", "May", "Jun",`<br>`  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec",`<br>`}` | ← *store predefined constants* |

# Arrays: quiz 1

**What are the contents of the array `a[]` after the loop terminates?**

**A.** A B C D E

**B.** A B C B A

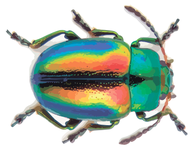
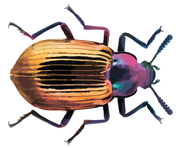
**C.** E D C B A

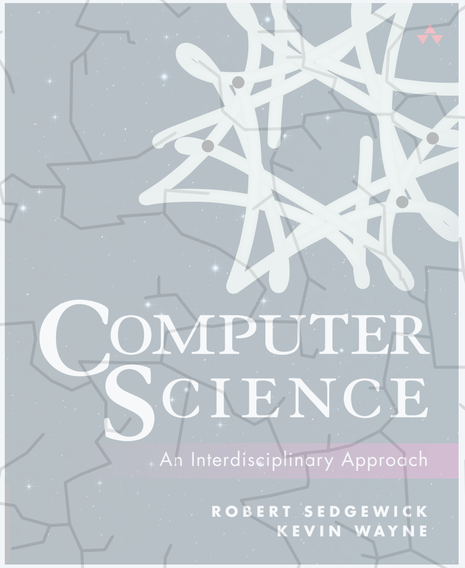
**D.** E D C D E

```
String[] a = { "A", "B", "C", "D", "E" };
int n = a.length;
for (int i = 0; i < n; i++) {
   String temp = a[i];
   a[i] = a[n-i-1];
   a[n-i-1] = temp;
}
```

*swap idiom*

# Programming with arrays: common bugs

| bug | buggy code | error | error message |
|---|---|---|---|
|  | `double[] a;`<br>`for (int i = 0; i < 10; i++)`<br>`  a[i] = Math.random();` | *uninitialized array*<br>*(compile-time error)* | `~/cos126/arrays> javac ArrayBug1.java`<br>`ArrayBug1.java:5: error:`<br>`variable a might not have`<br>`been initialized`<br>`  a[i] = Math.random();`<br>`  ^`<br>`1 error` |
|  | `double[] a = new int[10];`<br>`for (int i = 0; i < 10; i++)`<br>`  a[i] = Math.random();` | *type mismatch error*<br>*(compile-time error)* | `~/cos126/arrays> javac ArrayBug2.java`<br>`ArrayBug2.java:3: error:`<br>`incompatible types: int[]`<br>`cannot be converted to double[]`<br>`  double[] a = new int[10];`<br>`  ^`<br>`1 error` |
|  | `double[] a = new double[10];`<br>`for (int i = 1; i <= 10; i++)`<br>`  a[i] = Math.random();` | *array index out of bounds*<br>*(run-time error)* | `~/cos126/arrays> javac ArrayBug3.java`<br>`~/cos126/arrays> java ArrayBug3`<br>`Exception in thread "main"`<br>`java.lang.ArrayIndexOutOfBoundsException:`<br>`Index 10 out of bounds for length 1`<br>`  at ArrayBug3.java:5)` |




https://introcs.cs.princeton.edu

# 1.4 ARRAYS

- *basic concepts*
- *shuffling*
- *digital audio*
- *memory representation*
- *two-dimensional arrays*

# Create a deck of cards

Define three arrays:

- Ranks. `String[] ranks = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };`
- Suits. `String[] suits = { "♣", "♦", "♥", "♠" };`
- Full deck. `String[] deck = new String[52];`

Use nested for loops to put all cards in the deck.

```
for (int j = 0; j < 4; j++)
   for (int i = 0; i < 13; i++)
      deck[i + 13*j] = ranks[i] + suits[j];
```

suits[]

| j | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| | ♣ | ♦ | ♥ | ♠ |

ranks[]

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | J | Q | K | A |

deck[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2♣ | 3♣ | 4♣ | 5♣ | 6♣ | 7♣ | 8♣ | 9♣ | 10♣ | J♣ | Q♣ | K♣ | A♣ | 2♦ | 3♦ | 4♦ | 5♦ | 6♦ | 7♦ | 8♦ | 9♦ | … |

# Create a deck of cards

```
public class Deck {
   public static void main(String[] args) {
      String[] ranks = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
      String[] suits = { "♣", "♦", "♥", "♠" };

      String[] deck = new String[52];
      for (int j = 0; j < 4; j++)
         for (int i = 0; i < 13; i++)
            deck[i + 13*j] = ranks[i] + suits[j];

      for (int i = 0; i < 52; i++)
         System.out.print(deck[i] + " ");
       System.out.println();

   }
}
```

```
~/cos126/arrays> java Deck
2♣ 3♣ 4♣ 5♣ 6♣ 7♣ 8♣ 9♣ 10♣ J♣ Q♣ K♣ A♣ 2♦ 3♦ 4♦ 5♦ … 2♠ 3♠ 4♠ 5♠ 6♠ 7♠ 8♠ 9♠ 10♠ J♠ Q♠ K♠ A♠
```

# Arrays: quiz 2

**Which code fragment puts the cards in the array in rank order?**

```
~/cos126/arrays> java Deck
2♣ 2♦ 2♥ 2♠ 3♣ 3♦ 3♥ 3♠ 4♣ 4♦ 4♥ 4♠ 5♣ 5♦ 5♥ 5♠ … Q♣ Q♦ Q♥ Q♠ K♣ K♦ K♥ K♠ A♣ A♦ A♥
```

**A.**

```
for (int i = 0; i < 13; i++)
   for (int j = 0; j < 4; j++)
      deck[i + 13*j] = rank[i] + suit[j];
```

**B.**

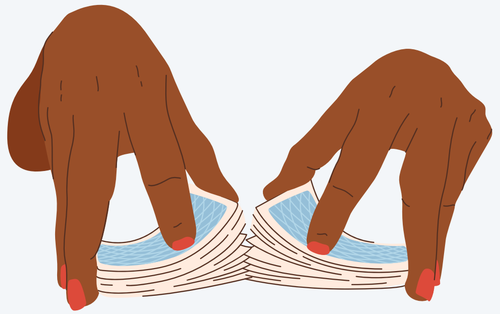
```
for (int i = 0; i < 13; i++)
   for (int j = 0; j < 4; j++)
      deck[4*i + j] = rank[i] + suit[j];
```

**C.** Both A and B.

**D.** Neither A nor B.

# Shuffling

Goal. Rearrange deck of cards in uniformly random order.



Algorithm. For each index $i$ from $0$ to $51$ :

- Pick a uniformly random index $r$ between $0$ and $i$.
- Exchange `deck[i]` and `deck[r]`.

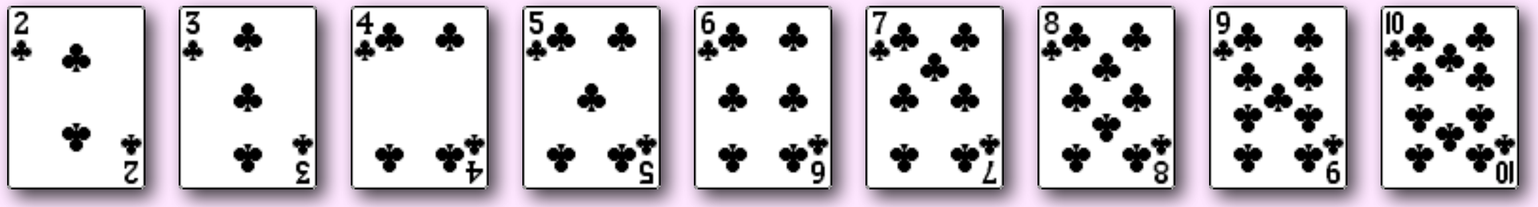
```
for (int i = 0; i < 52; i++) {
   int r = (int) (Math.random() * (i+1));
   String temp = deck[r];
   deck[r] = deck[i];
   deck[i] = temp;
}
```

*between* `0` *and* `i` (*equally likely*)

# Shuffling demo

Algorithm. For each index $i$ from $0$ to $n-1$ :

- Pick a uniformly random index $r$ between $0$ and $i$.
- Exchange `a[i]` and `a[r]`.



# Shuffling trace

```
for (int i = 0; i < 9; i++) {
   int r = 1 + (int) (Math.random() * (i+1));
   String temp = deck[r];
   deck[r] = deck[i];
   deck[i] = temp;
}
```

| | | deck[] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| i | r | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | | 2♣ | 3♣ | 4♣ | 5♣ | 6♣ | 7♣ | 8♣ | 9♣ | 10♣ |
| 0 | 0 | 2♣ | 3♣ | 4♣ | 5♣ | 6♣ | 7♣ | 8♣ | 9♣ | 10♣ |
| 1 | 0 | 3♣ | 2♣ | 4♣ | 5♣ | 6♣ | 7♣ | 8♣ | 9♣ | 10♣ |
| 2 | 0 | 4♣ | 2♣ | 3♣ | 5♣ | 6♣ | 7♣ | 8♣ | 9♣ | 10♣ |
| 3 | 2 | 4♣ | 2♣ | 5♣ | 3♣ | 6♣ | 7♣ | 8♣ | 9♣ | 10♣ |
| 4 | 1 | 4♣ | 6♣ | 5♣ | 3♣ | 2♣ | 7♣ | 8♣ | 9♣ | 10♣ |
| 5 | 0 | 7♣ | 6♣ | 5♣ | 3♣ | 2♣ | 4♣ | 8♣ | 9♣ | 10♣ |
| 6 | 4 | 7♣ | 6♣ | 5♣ | 3♣ | 8♣ | 4♣ | 2♣ | 9♣ | 10♣ |
| 7 | 7 | 7♣ | 6♣ | 5♣ | 3♣ | 8♣ | 4♣ | 2♣ | 9♣ | 10♣ |
| 8 | 1 | 7♣ | 10♣ | 5♣ | 3♣ | 8♣ | 4♣ | 2♣ | 9♣ | 6♣ |
| | | 7♣ | 10♣ | 5♣ | 3♣ | 8♣ | 4♣ | 2♣ | 9♣ | 6♣ |

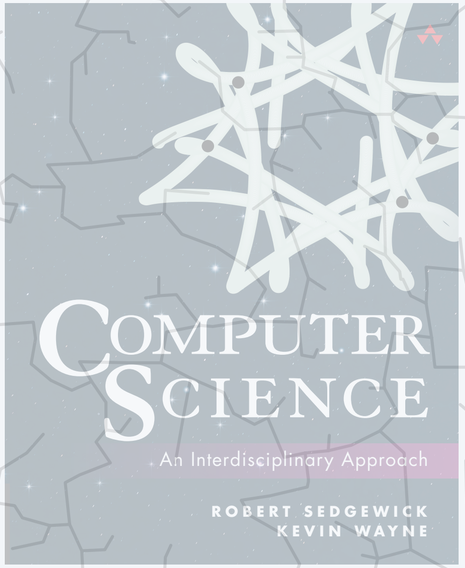
**trace of variables (at end of each iteration)**

# Shuffling a deck of cards: implementation

```
public class ShuffledDeck {
   public static void main(String[] args) {
      String[] ranks = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
      String[] suits = { "♣", "♦", "♥", "♠" };
      int RANKS = ranks.length;
      int SUITS = suits.length;
      int n = RANKS * SUITS;

      String[] deck = new String[n];                    create deck
      for (int j = 0; j < SUITS; j++)
         for (int i = 0; i < RANKS; i++)
            deck[i + RANKS*j] = ranks[i] + suits[j];

      for (int i = 0; i < n; i++) {
         int r = (int) (Math.random() * (i+1));
         String temp = deck[r];
         deck[r] = deck[i];                             shuffle deck
         deck[i] = temp;
      }

      for (int i = 0; i < n; i++)                       print deck
         System.out.print(deck[i] + " ");

   }
}
```

*avoid "magic constants" (such as 4, 13, and 52)*

```
~/cos126/arrays> java ShuffledDeck
8♠ A♦ A♥ 9♦ 6♥ 7♥ 9♠ Q♥ … K♣ 2♣ 6♦ 2♦ 5♥

~/cos126/arrays> java ShuffledDeck
K♦ J♥ 7♦ 9♦ Q♦ 5♥ 6♥ 9♥ … Q♥ K♠ 4♦ 6♠ 7♣
```



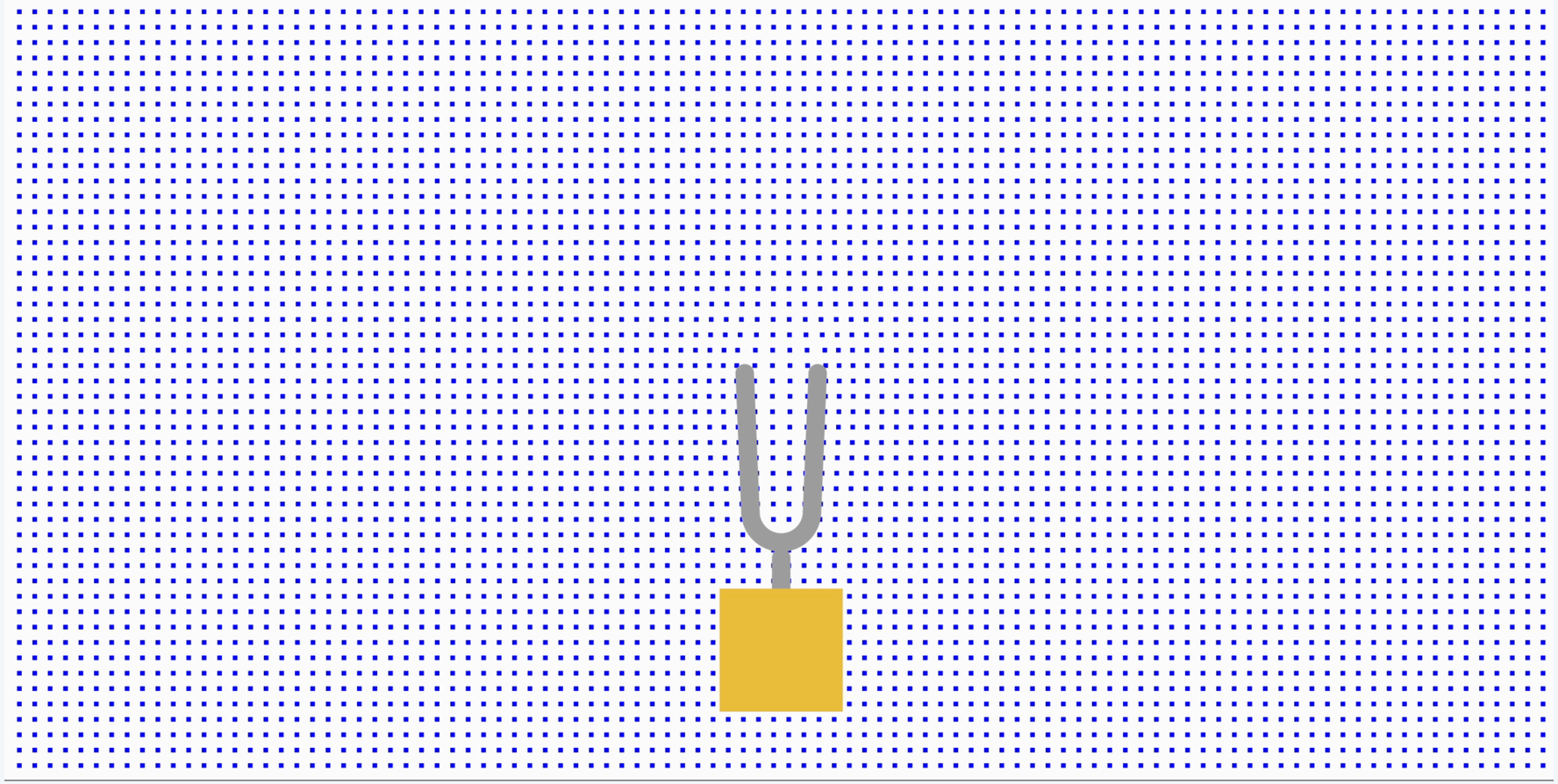

https://introcs.cs.princeton.edu

# 1.4 ARRAYS

- *basic concepts*
- *shuffling*
- *digital audio*
- *memory representation*
- *two-dimensional arrays*

# Crash course in sound

Sound. The perceptible vibration of air by the ear.



https://javalab.org/en/tuning_fork_and_sound_wave_en

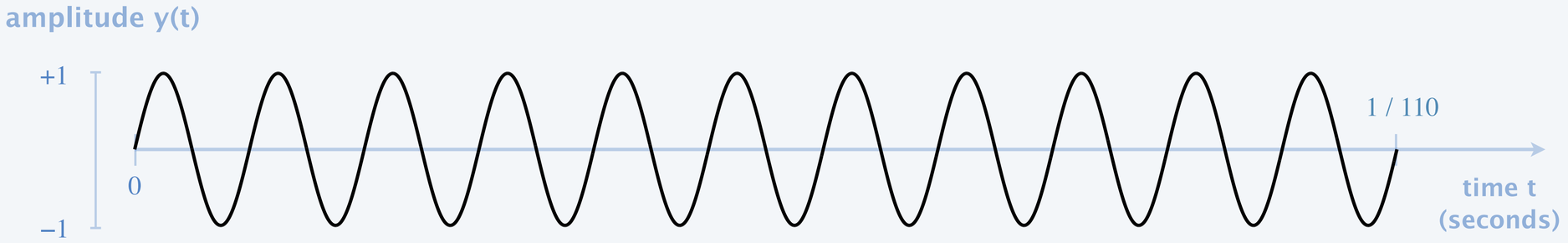
# Crash course in digital audio

**Audio.** An analog or digital encoding of sound.

**Audio formats.** Vinyl, tape cassette, CD, WAV, MP3, AIFC, …

**Audio signal.** Real-valued (between −1 and +1) function of time. ← *value (amplitude) relates to change in sound pressure*

- A loudspeaker converts an audio signal into sound.
- A microphone converts sound into an audio signal.



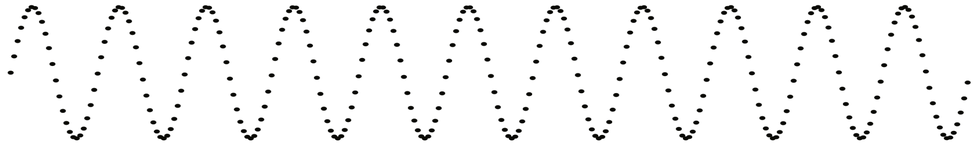
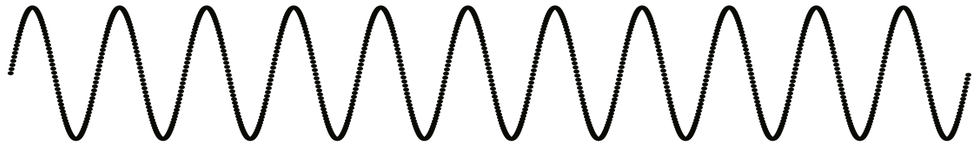

**1/110 second of concert A**
**(sine wave with frequency 440 Hz)**

# Audio sampling

Goal. Convert a continuous-time signal into a discrete-time signal.

- A sample is a signal value at specific point in time.
- Take samples at evenly spaced points.

*model sound with an array of real numbers between −1 and +1 (using 44,100 samples per second)*

| samples / second | samples | samples from a sine wave (440 Hz) |
|---|---|---|
| 5,512 | 138 |  |
| 11,025 | 276 |  |
| 22,050 | 552 |  |
| 44,100 | 1103 |  |

*a standard sampling rate*

1/40 second of concert A

# Standard audio library

*StdAudio*. Our library for playing, reading, and saving digital audio.

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

`public class StdAudio`

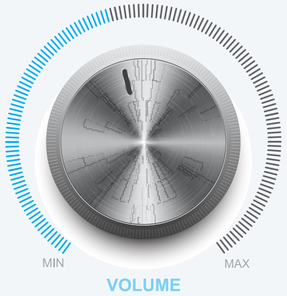
| | | |
|---|---|---|
| `static int` | `SAMPLE_RATE` | 44,100 (*CD quality audio*) |
| `static void` | `play(String filename)` | *play the audio file* |
| `static void` | `playInBackground(String filename)` | *play the audio file in the background* |
| `static void` | `play(double sample)` | *play the sample* |
| `static void` | `play(double[] samples)` | *play the samples* |
| `static double[]` | `read(String filename)` | *read the samples from an audio file* |
| `static void` | `save(String filename, double[] samples)` | *save the samples to an audio file* |
| `static void` | `drain()` | *play any samples left in buffer* |
| ⋮ | | ⋮ |

*1 hour of audio comprises about* 159 *million samples*

*supported file formats: WAV, AU, AIFF, MIDI*

# Audio gain

**Volume.** Perceived loudness of a sound.




**Audio gain.** Multiply all samples by the same constant $\alpha$.

- $|\alpha| > 1 \Rightarrow$ amplifies audio signal.
- $|\alpha| < 1 \Rightarrow$ attenuates audio signal.

```
public class Gain {
   public static void main(String[] args) {
      double[] samples = StdAudio.read(args[0]);
      double alpha = Double.parseDouble(args[1]);

      for (int i = 0; i < samples.length; i++) {
         samples[i] *= alpha;
         if (samples[i] > +1.0) samples[i] = +1.0;
         if (samples[i] < -1.0) samples[i] = -1.0;
      }

      StdAudio.play(samples);
   }
}
```

*"clipping"*

```
~/cos126/arrays> java-introcs Gain Game.wav 1.0
[plays sound effect]

~/cos126/arrays> java-introcs Gain Game.wav 2.0
[plays louder version]

~/cos126/arrays> java-introcs Gain Game.wav 0.5
[plays quieter version]

~/cos126/arrays> java-introcs Gain Game.wav 0.0
[plays silence]

~/cos126/arrays> java-introcs Gain Game.wav -1.0
[plays inverted version]
```

# Arrays: quiz 3

**What sound will the following command produce?**

**A.** Original audio.

**B.** Silence.

**C.** Static.

**D.** Ear-shattering noise.

**E.** None of the above.

```
~/cos126/arrays> java-introcs Gain HelloWorld.wav 9999.99
[plays sound with ???]
```

```
double[] samples = StdAudio.read("HelloWorld.wav");
for (int i = 0; i < samples.length; i++) {
   if      (samples[i] < 0.0) samples[i] = -1.0;
   else if (samples[i] > 0.0) samples[i] =  1.0;
}
StdAudio.play(samples);
```

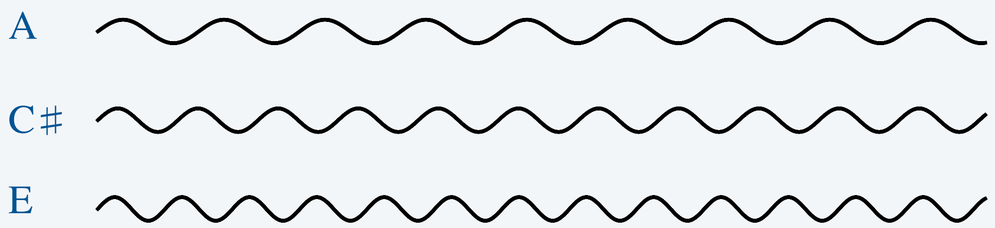
**effectively equivalent to**

# Principle of superposition

Superposition. To combine two (or more) audio signals, add the corresponding samples.

*sound waves are mechanical waves*

Ex 1. Add audio signals of notes to produce a chord.



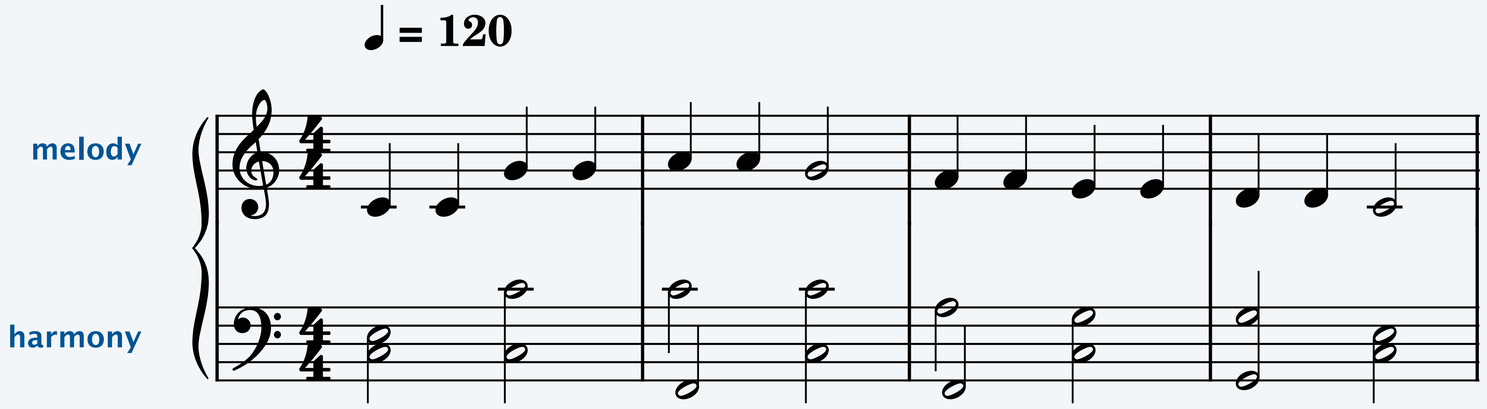

*A major chord*

# Principle of superposition

Superposition. To combine two (or more) audio signals, add the corresponding samples.

Ex 1. Add audio signals of notes to produce a chord.

Ex 2. Add audio signals of parts, instruments, and voices to produce a musical composition.




**“Twinkle, Twinkle, Little Star”**
**(two parts)**

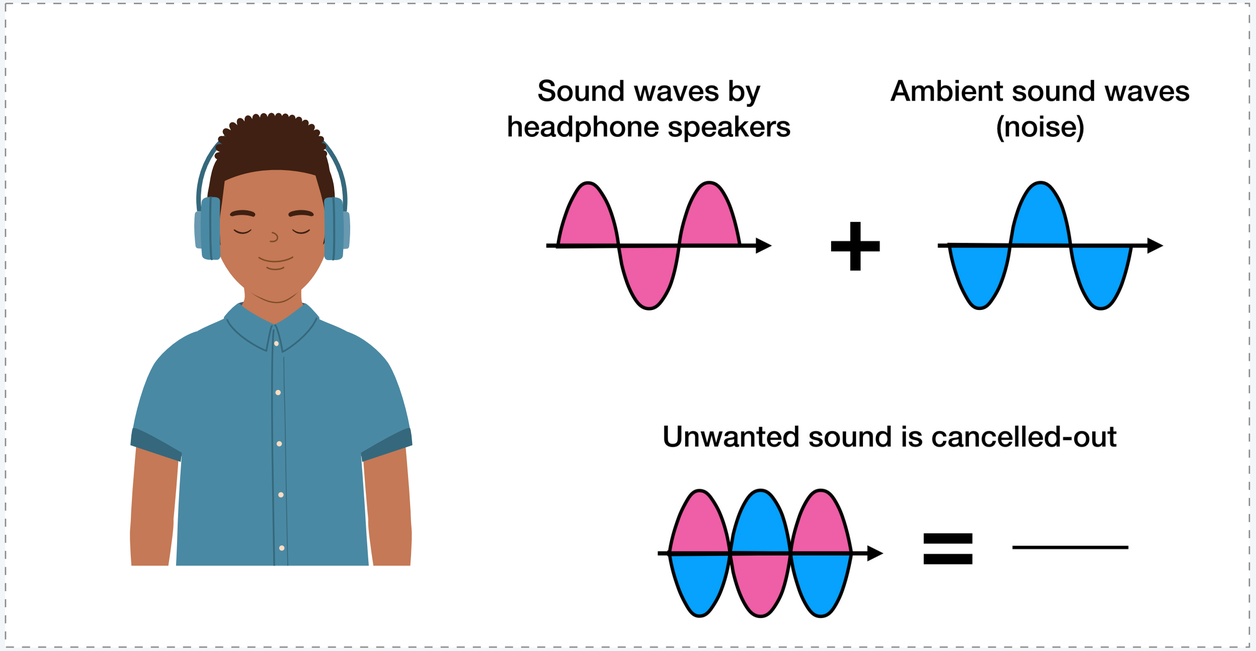
# Principle of superposition

Superposition. To combine two (or more) audio signals, add the corresponding samples.

Ex 1. Add audio signals of notes to produce a chord.

Ex 2. Add audio signals of parts, instruments, and voices to produce a musical composition.

Ex 3. Noise-cancelling headphones.



# Superposition of audio files

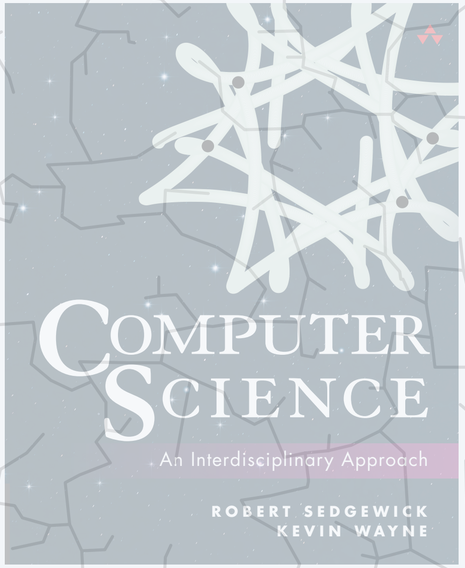
```
public class Superpose {
   public static void main(String[] args) {
      double[] results = StdAudio.read(args[0]);
      for (int i = 1; i < args.length; i++) {
      double[] samples = StdAudio.read(args[i]);
         for (int j = 0; j < samples.length; j++) {
            results[j] += samples[j];
         }
      }

      StdAudio.play(results);
   }
}
```

args[] *in* main() *is a* String *array*

*use a loop to add the corresponding samples (assumes all arrays of same length)*

*play the results*

```
~/cos126/arrays> java-introcs Superpose PacManMelody.wav
[plays Pac-Man startup melody]

~/cos126/arrays> java-introcs Superpose PacManHarmony.wav
[plays Pac-Man startup harmony]

~/cos126/arrays> java-introcs Superpose PacManMelody.wav PacManHarmony.wav
[plays Pac-Man startup melody and harmony]
```




https://introcs.cs.princeton.edu

# 1.4 ARRAYS

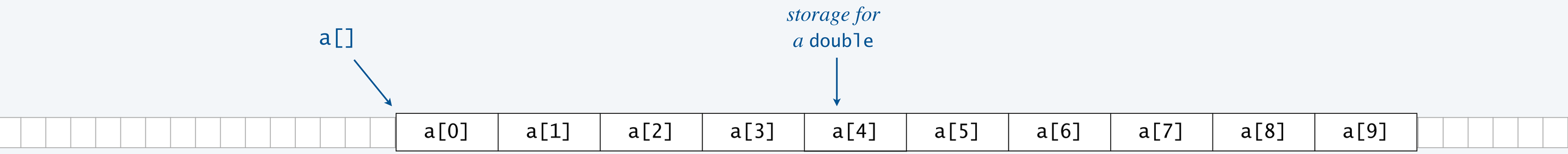
- *basic concepts*
- *shuffling*
- *digital audio*
- *memory representation*
- *two-dimensional arrays*

# Memory representation of an array

Java array. An array is an indexed sequence of values of the same type.

Computer memory. Your computer's memory is an indexed sequence of memory locations.

- Each *int*, *double*, or *boolean* occupies a fixed number of memory locations.
- Array elements are stored in contiguous memory locations.




Key properties.

- Given index `i`, accessing `a[i]` is extremely efficient.
- Once you create an array, you can never change its type or length.
- Arrays are reference types, not primitive types.

*think of the variable* `a[]` *as storing the memory address of its first element*

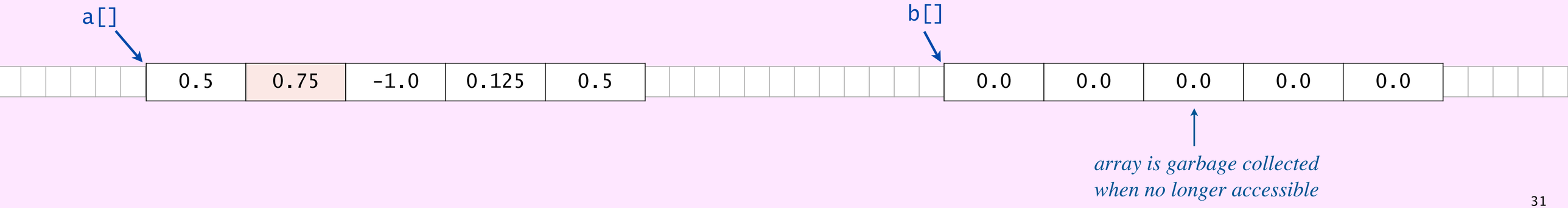
# Assignment statements with arrays

Consequence 1. The assignment statement *b = a* makes *a* and *b* refer to the same array.

*it does not create a new, independent, array*

Ex.

```
double[] a = { 0.5, 0.25, -1.0, 0.125, 0.5 };
double[] b = new double[a.length];
b = a;
b[1] = 0.75;
```



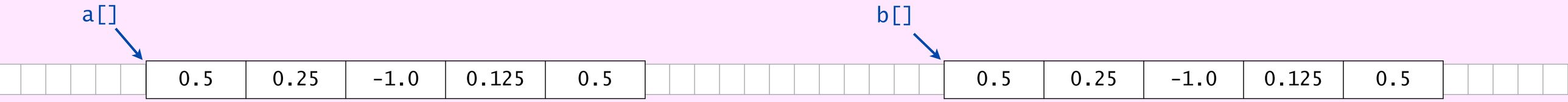
# Checking arrays for equality

Consequence 2. The expression *a == b* checks whether *a* and *b* refer to the same array.

*not whether they store the same sequence of values*

Ex.

```
double[] a = { 0.5, 0.25, -1.0, 0.125, 0.5 };
double[] b = { 0.5, 0.25, -1.0, 0.125, 0.5 };
System.out.println(a == b);   // false
```



# Copying an array and checking for equality

Q. How to copy an array and check for equality?

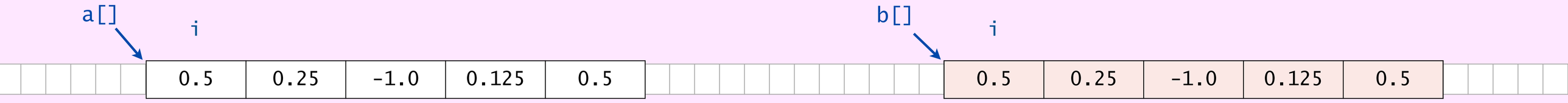
A. Use loops.

```
double[] a = { 0.5, 0.25, -1.0, 0.125, 0.5 };
double[] b = new double[a.length];
for (int i = 0; i < a.length; i++)
    b[i] = a[i];
```

**copying an array**

```
boolean areEqual = true;
for (int i = 0; i < a.length; i++) {
   if (a[i] != b[i])
      areEqual = false;
}
```

**checking two arrays (of same length) for equality**



# Arrays: quiz 4

**What does the following code fragment print?**

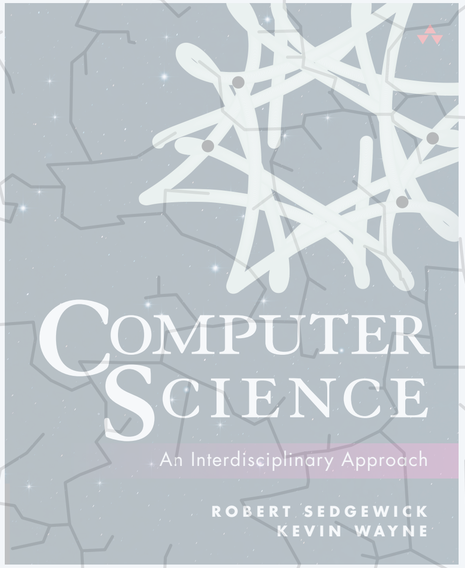
**A.** 0 1 2 0 1 2

**B.** 0 1 2 1 2 6

**C.** 1 2 6 0 1 2

**D.** 1 2 6 1 2 6

```
int[] a = { 1, 2, 6 };
int[] b = new int[a.length];

b = a;
for (int i = 0; i < b.length; i++)
   b[i] = i;

for (int i = 0; i < a.length; i++)
   System.out.print(a[i] + " ");

for (int i = 0; i < b.length; i++)
   System.out.print(b[i] + " ");
```




https://introcs.cs.princeton.edu

# 1.4 ARRAYS

- *basic concepts*
- *shuffling*
- *digital audio*
- *memory representation*
- *two-dimensional arrays*

# Two-dimensional arrays

A two-dimensional array is a *doubly-indexed* table of values of the same type.

Examples.

- Grades for students in an online class.
- Outcomes of a scientific experiment.
- Customer transactions in a bank.
- Entries in a feature matrix.
- Pixels in a digital image.
- Cells in a spreadsheet.
- …

*grade*

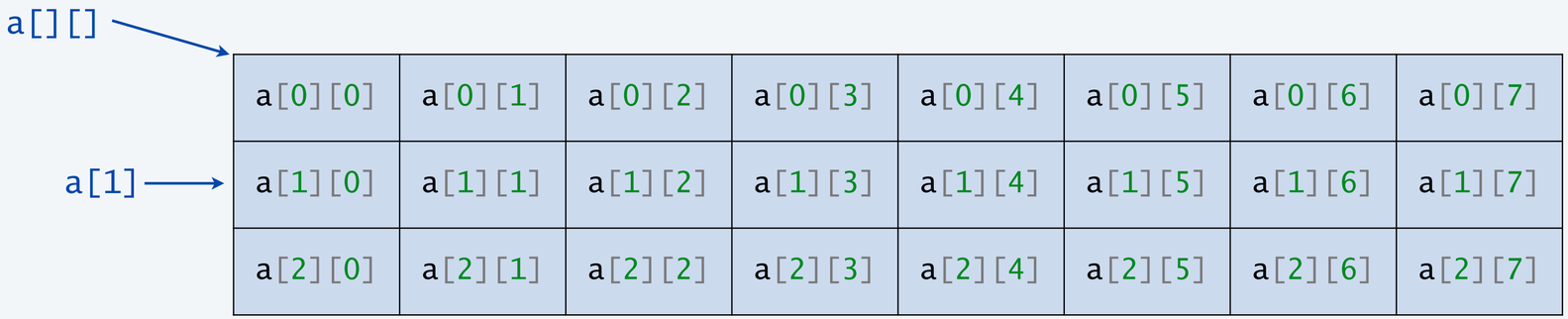
| *student ID* | 0 | 1 | 2 | 3 | 4 | 5 | ⋯ |
|---|---|---|---|---|---|---|---|
| 0 | A | A | C | B | A | C | |
| 1 | B | B+ | B | B− | A | A− | |
| 2 | C | D | D | B | C | A | |
| 3 | A | A+ | A | A− | A | A+ | |
| 4 | C | C | B+ | C | B | B− | |
| ⋮ | | | | | | | |

# Two-dimensional arrays in Java

| operation | typical code |
|---|---|
| *declare a two-dimensional array* | `double[][] a;` |
| *create an m-by-n array* | `a = new double[m][n];` |
| *declare, create, and initialize in one statement* | `double[][] a = new double[m][n];` |
| *refer to an array element by index* | `a[i][j] = b[i][j] + c[j][k];` |
| *number of rows* | `a.length` |
| *number of columns* | `a[i].length` |

*all elements initialized to default value (zero for numeric types, false for* `boolean`*)*

*can be different for each row ("ragged" array)*



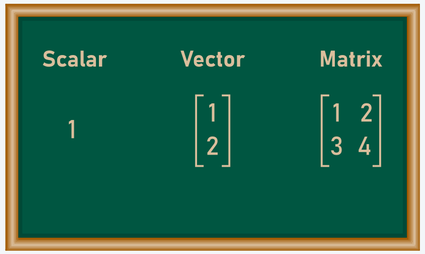

*same conventions as matrices*

**a 3-by-8 array**

# Vector and matrix calculations

Mathematical abstractions. Vectors and matrices.

Java implementation. 1D arrays and 2D arrays.




**vector addition**

```
double[] c = new double[n];
for (int i = 0; i < n; i++)
   c[i] = a[i] + b[i];
```

**matrix addition**

```
double[][] c = new double[n][n];
for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      c[i][j] = a[i][j] + b[i][j];
```

$$\underbrace{(0.8,\ 0.7,\ 0.5)}_{c} = \underbrace{(0.3,\ 0.6,\ 0.1)}_{a} + \underbrace{(0.5,\ 0.1,\ 0.4)}_{b}$$

$$\underbrace{\begin{bmatrix} 1.5 & 0.5 & 0.6 \\ 0.4 & 1.0 & 0.2 \\ 0.6 & 0.4 & 0.8 \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.3 & 0.6 & 0.1 \\ 0.5 & 0.1 & 0.4 \end{bmatrix}}_{A} + \underbrace{\begin{bmatrix} 0.8 & 0.3 & 0.5 \\ 0.1 & 0.4 & 0.1 \\ 0.1 & 0.3 & 0.4 \end{bmatrix}}_{B}$$

# Vector and matrix calculations

Mathematical abstractions. Vectors and matrices.

Java implementation. 1D arrays and 2D arrays.

**vector dot product**

```
double sum = 0.0;
for (int i = 0; i < n; i++)
   sum += a[i] * b[i];
```

$$0.25 = \underbrace{(0.3, 0.6, 0.1)}_{a} \cdot \underbrace{(0.5, 0.1, 0.4)}_{b}$$

| i | a[i] | b[i] | a[i]*b[i] | sum |
|---|---|---|---|---|
| 0 | 0.3 | 0.5 | 0.15 | 0.15 |
| 1 | 0.6 | 0.1 | 0.06 | 0.21 |
| 2 | 0.1 | 0.4 | 0.04 | 0.25 |

**matrix multiplication**

```
double[][] c = new double[n][n];
for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
         c[i][j] += a[i][k] * b[k][j];
```

$$\underbrace{\begin{bmatrix} 0.59 & 0.32 & 0.41 \\ 0.31 & 0.36 & 0.25 \\ 0.45 & 0.31 & 0.42 \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.3 & 0.6 & 0.1 \\ 0.5 & 0.1 & 0.4 \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} 0.8 & 0.3 & 0.5 \\ 0.1 & 0.4 & 0.1 \\ 0.1 & 0.3 & 0.4 \end{bmatrix}}_{B}$$

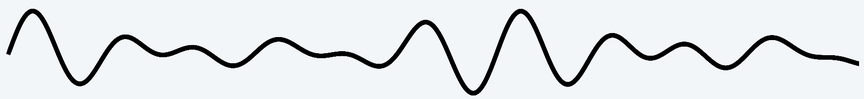
# Summary

An array is an *indexed sequence* of values of the same type.

- Serves as a basic building block in programming.
- Enables efficient manipulation of large amounts of data.

Some examples. [in this course]

**digital audio signal**

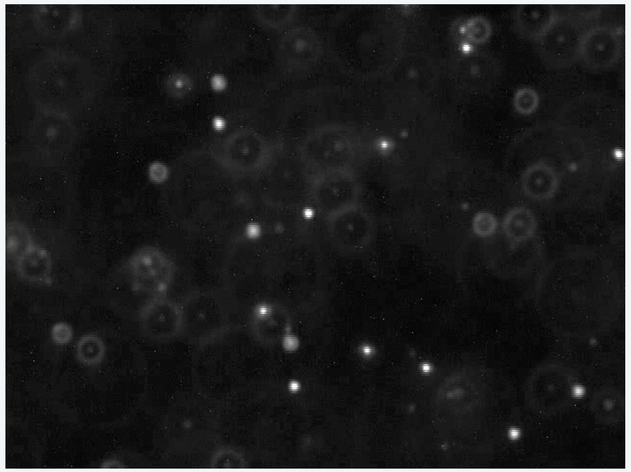


**DNA string**

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

**digital image**

**digital video**



# Credits

| media | source | license |
|---|---|---|
| *Johnson Arch* | Danielle Alio Capparella | by photographer |
| *DNA* | Adobe Stock | education license |
| *CERN Server* | Florian Hirzinger | CC BY-SA 3.0 |
| *Fanned Cards* | clipart-library.com | non-commercial use |
| *Bugs* | Adobe Stock | education license |
| *Deck of Cards* | Adobe Stock | education license |
| *Card Shuffling* | Adobe Stock | education license |
| *Sound Wave Set* | Adobe Stock | education license |
| *Tuning Fork and Sound Wave* | Javalab | |
| *Ear Listening* | Adobe Stock | education license |
| *Tuning Fork Sound Effect* | Pixabay | Pixabay content license |

# Credits

| media | source | license |
|---|---|---|
| *Retro Microphone* | Adobe Stock | education license |
| *Headphones* | Adobe Stock | education license |
| *Volume Control* | Adobe Stock | education license |
| *Noise Cancellation* | Adobe Stock | education license |
| *Boy with Headphones* | Adobe Stock | education license |
| *Pac-Man Startup Sound* | Bandai Namco Entertainment | |
| *Crane Song* | Matthew White | public domain |
| *Poker Face* | Lady Gaga | |
| *Scalar, Vector, and Matrix* | Adobe Stock | education license |
| *Mandrill* | USC SIPI Image Database | |
| *Johnson Arch* | Danielle Alio Capparella | by photographer |