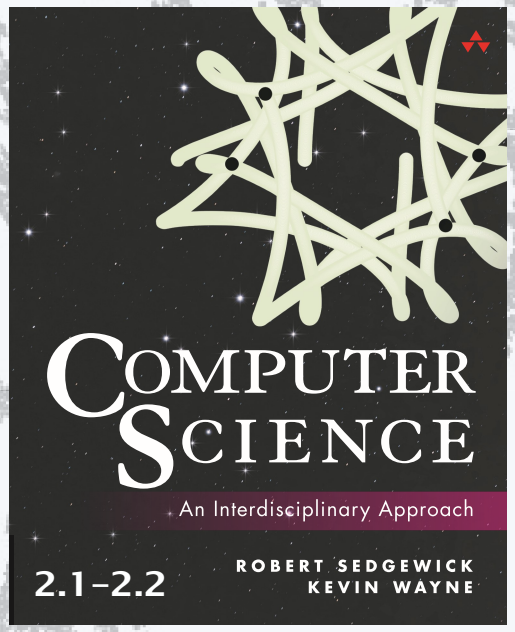
COMPUTER SCIENCE

SEDGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA



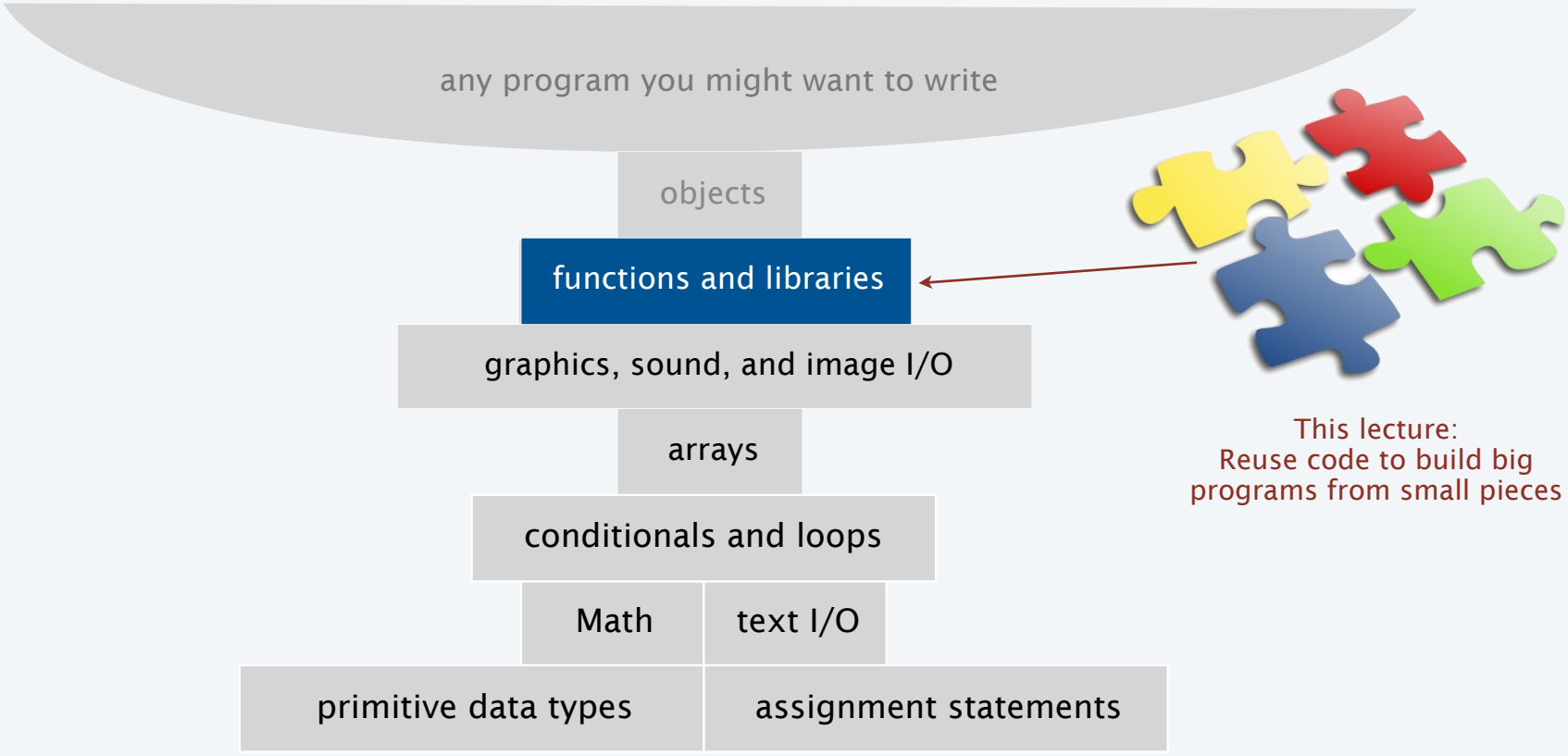
http://introcs.cs.princeton.edu

5. Functions and Libraries

5. Functions and Libraries

- Basic concepts
- Case study: Digital audio
- Application: Gaussian distribution
- Modular programming and libraries

Context: basic building blocks for programming



Functions, libraries, and modules

Modular programming

- Organize programs as independent modules that do a job together.
- Why? Easier to share and reuse code to build bigger programs.

Facts of life

- Support of modular programming has been a holy grail for decades.
- Ideas can conflict and get highly technical in the real world.

Def. A library is a set of functions.

↑ for purposes of this lecture

Def. A module is a `.java` file.

↑ for purposes of this course

For now. Libraries and modules are the *same thing*: `.java` files containing sets of functions.

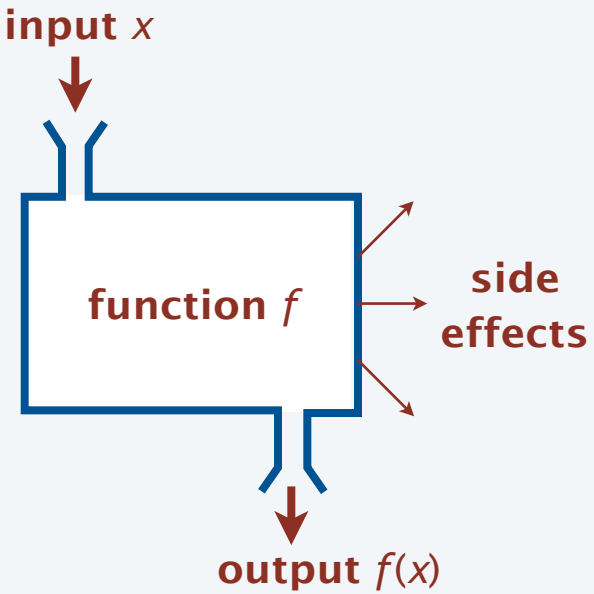
Later. Modules implement *data types* (stay tuned).

Functions (static methods)

Java *function* ("aka *static method*")

- Takes zero or more *input* arguments.
- Returns zero or one *output* value.
- May cause *side effects* (e.g., output to standard draw).

Java functions are *more general* than mathematical functions



Applications

- Scientists use mathematical functions to calculate formulas.
- Programmers use functions to build modular programs.
- *You* use functions for both.

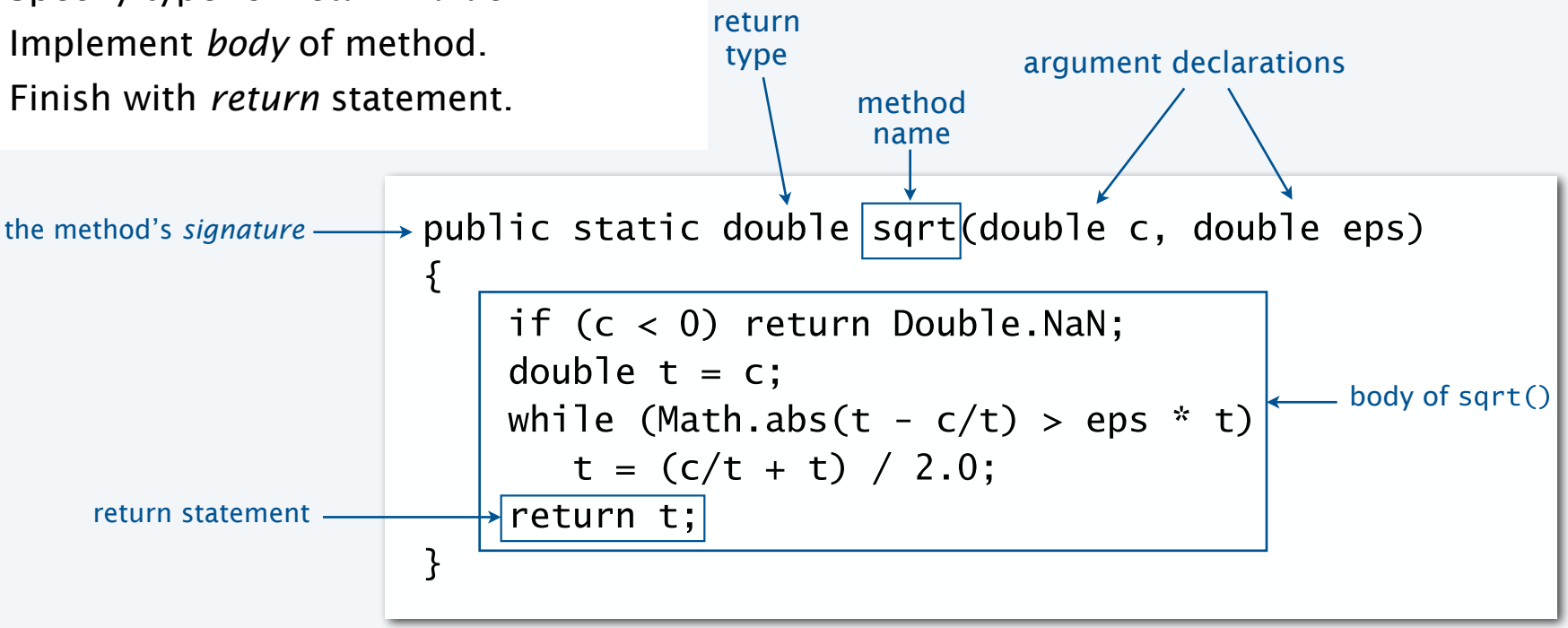
Examples seen so far

- Built-in functions: `Math.random()`, `Math.abs()`, `Integer.parseInt()`.
- Our I/O libraries: `StdIn.readInt()`, `StdDraw.line()`, `StdAudio.play()`.
- User-defined functions: `main()`.

Anatomy of a Java static method

To implement a function (static method)

- Create a *name*.
- Declare type and name of *argument*(*s*).
- Specify type for *return value*.
- Implement *body* of method.
- Finish with *return* statement.



Anatomy of a Java library

A library is a set of functions.

Note: We are using our `sqrt()` from Lecture 2 here to illustrate the basics with a familiar function.

Our focus is on control flow here. See Lecture 2 for technical details.

You can use `Math.sqrt()`.

sqrt() method

module named Newton.java

main() method

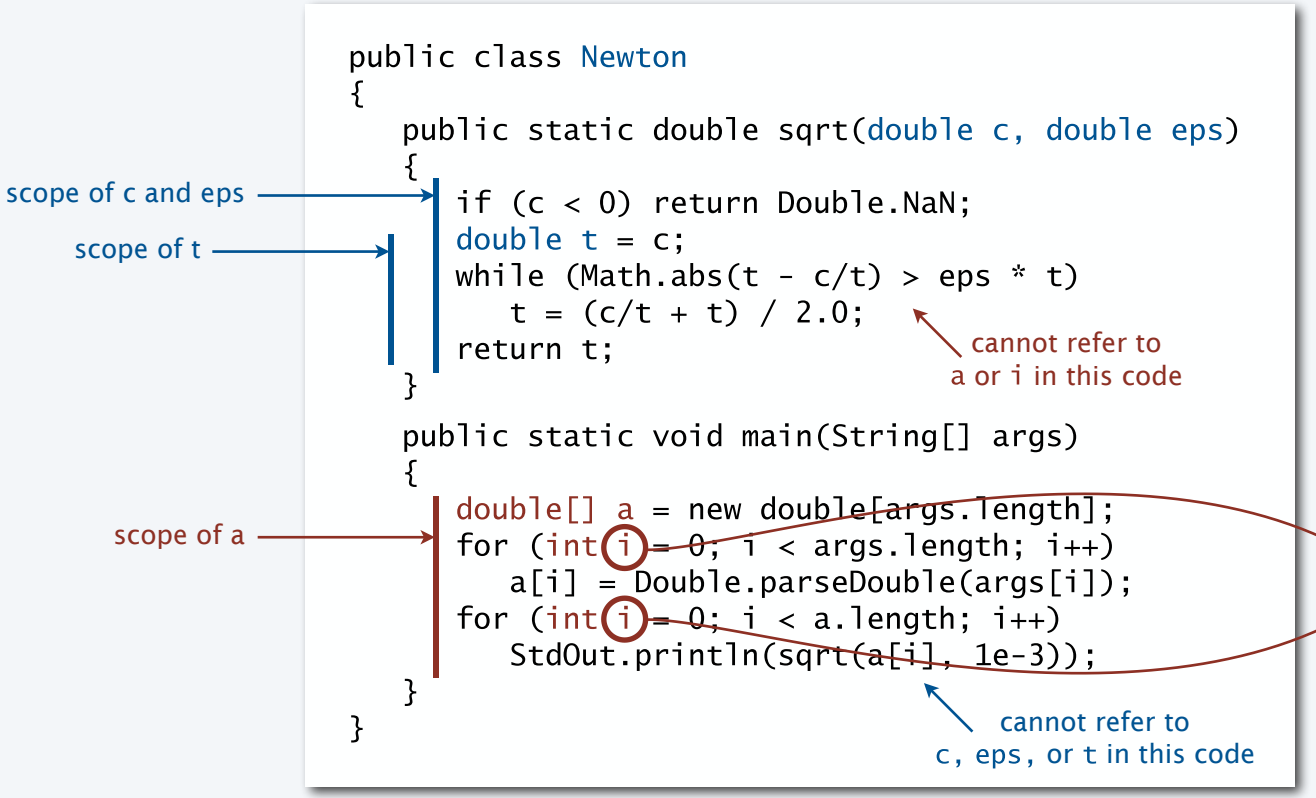
```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
         StdOut.println(sqrt(a[i], 1e-3));
   }
}
```

library/module name

Key point. Functions provide a *new way* to control the flow of execution.

Scope

Def. The scope of a variable is the code that can refer to it by name.



In a Java library, a variable's scope is the code following its declaration, in the same block.

Best practice. Declare variables so as to *limit* their scope.

Flow of control

```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * t)
         t = (c/t + t) / 2.0;
      return t;
   }
   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
      {
         double x = sqrt(a[i], 1e-3);
         StdOut.println(x);
      }
   }
}
```

Summary of flow control for a function call

- Control transfers to the function code.
- Argument variables are declared and initialized with the given values.
- Function code is executed.
- Control transfers back to the calling code (with return value assigned in place of the function name in the calling code).

"pass by value"
(other methods used in other systems)

Note. OS calls `main()` on `java` command

Function call flow of control trace

```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * t)
         t = (c/t + t) / 2.0;
      return t;
   }
   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
      {
         double x = sqrt(a[i], 1e-3);
         StdOut.println(x);
      }
   }
}
```

c	t
3.0	3.0
	2.0
	1.75
	1.732

i	a[i]	x
0	1.0	1.000
1	2.0	1.414
2	3.0	1.732
3		

```
% java Newton 1 2 3
1.000
1.414
1.732
```

Pop quiz 1a on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1a
{
   public static int cube(int i)
   {
      int j = i * i * i;
      return j;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1a on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1a
{
   public static int cube(int i)
   {
      int j = i * i * i;
      return j;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Takes *N* from the command line, then prints cubes of integers from 1 to *N*

```
% javac PQfunctions1a.java
% java PQfunctions1a 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Pop quiz 1b on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1b
{
   public static int cube(int i)
   {
      int i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1b on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1b
{
   public static int cube( int i )
   {
      int i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Won't compile. Argument variable i is declared and initialized for function block, so the name cannot be reused.

```
% javac PQfunctions1b.java
PQfunctions1b.java:5: i is already defined in cube(int)
      int i = i * i * i;
          ^
1 error
```

Pop quiz 1c on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1c
{
   public static int cube(int i)
   {
      i = i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1c on functions

Q. What happens when you compile and run the following code?

```
public class PQ6_1c
{
   public static int cube(int i)
   {
      i = i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Won't compile. Need `return` statement.

```
% javac PQfunctions1c.java
PQfunctions1c.java:6: missing return statement
   }
   ^
1 error
```

Pop quiz 1d on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1d
{
   public static int cube(int i)
   {
      i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1d on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1d
{
   public static int cube(int i)
   {
      i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Works. The `i` in `cube()` is

- Declared and initialized as an argument.
- Different from the `i` in `main()`.

BUT changing values of function arguments is sufficiently confusing to be deemed bad style for this course.

```
% javac PQfunctions1d.java
% java PQfunctions1d 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Pop quiz 1e on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1e
{
   public static int cube(int i)
   {
      return i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1e on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1e
{
   public static int cube(int i)
   {
      return i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Works fine. Preferred (compact) code.

```
% javac PQfunctions1e.java
% java PQfunctions1e 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Image sources

http://upload.wikimedia.org/wikipedia/commons/b/ba/Working_Together_Teamwork_Puzzle_Concept.jpg

http://pixabay.com/en/ball-puzzle-pieces-of-the-puzzle-72374/

http://upload.wikimedia.org/wikipedia/commons/e/ef/Ben_Jigsaw_Puzzle_Puzzle_Puzzle.png

http://en.wikipedia.org/wiki/Function_(mathematics)#mediaviewer/File:Function_machine2.svg

5. Functions and Libraries

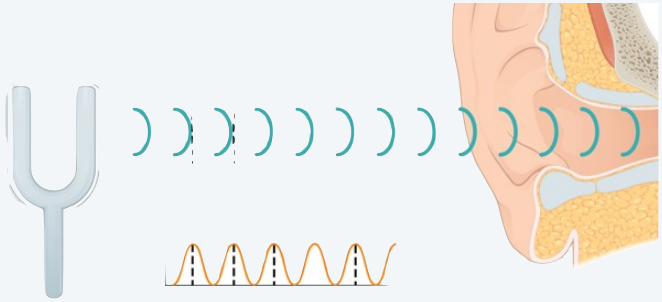
- Basic concepts
- Case study: Digital audio
- Application: Gaussian distribution
- Modular programming

Crash course in sound

Sound is the perception of the vibration of molecules.

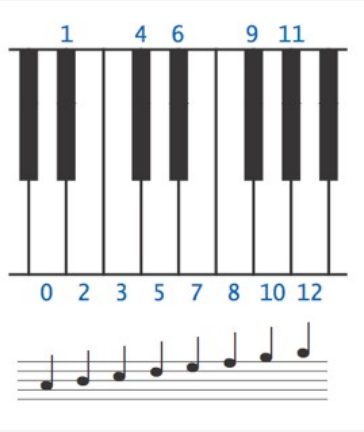
A musical tone is a steady periodic sound.

A pure tone is a sinusoidal waveform.



Western musical scale

- Concert A is 440 Hz.
- 12 notes, logarithmic scale.

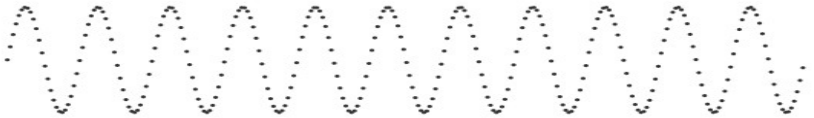
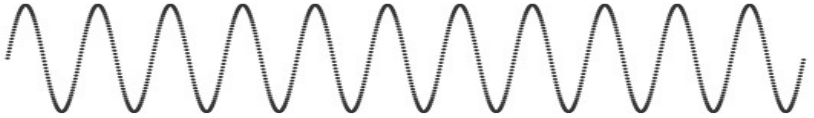


pitch	*i*	*frequency* ($440*2^{i/12}$)	*sinusodial waveform*
A	0	440	
A♯ / B♭	1	466.16	
B	2	493.88	
C	3	523.25	
C♯ / D♭	4	554.37	
D	5	587.33	
D♯ / E♭	6	622.25	
E	7	659.26	
F	8	698.46	
F♯ / G♭	9	739.99	
G	10	783.99	
G♯ / A♭	11	830.61	
A	12	880	

Digital audio

To represent a wave, *sample* at regular intervals and save the values in an array.

same as when plotting a function (previous lecture)

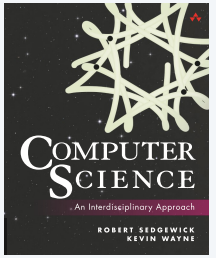
	samples/sec	*samples*	*sampled waveform*
1/40 second of concert A	5,512	137	
	11,025	275	
	22,050	551	
CD standard →	44,100	1102	

Bottom line. You can *write programs* to manipulate sound (arrays of double values).

StdAudio library

Developed for this course, also broadly useful

- Play a sound wave (array of double values) on your computer's audio output.
- Convert to and from standard `.wav` file format.



API

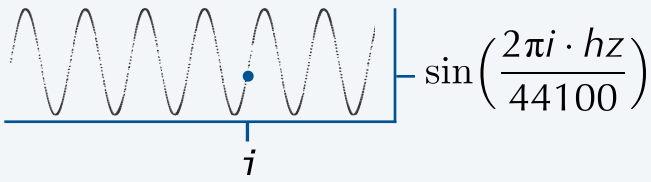
`public class StdAudio`	
`void play(String file)`	*play the given* `.wav` *file*
`void play(double[] a)`	*play the given sound wave*
`void play(double x)`	*play the sample for 1/44100 second*
`void save(String file, double[] a)`	*save to a* `.wav` *file*
`double[] read(String file)`	*read from a* `.wav` *file*

Enables you to *hear the results* of your programs that manipulate sound.

"Hello, World" for StdAudio

```
public class PlayThatNote
{
   public static double[] tone(double hz, double duration)
   {
      int N = (int) (44100 * duration);
      double[] a = new double[N+1];
      for (int i = 0; i <= N; i++)
         a[i] = Math.sin(2 * Math.PI * i * hz / 44100);
     return a;
   }

   public static void main(String[] args)
   {
      double hz = Double.parseDouble(args[0]);
      double duration = Double.parseDouble(args[1]);
      double[] a = tone(hz, duration);
      StdAudio.play(a);
   }
}
```

pitch




```
% java PlayThatNote 440.0 3.0
```

```
% java PlayThatNote 880.0 3.0
```

```
% java PlayThatNote 220.0 3.0
```

```
% java PlayThatNote 494.0 3.0
```

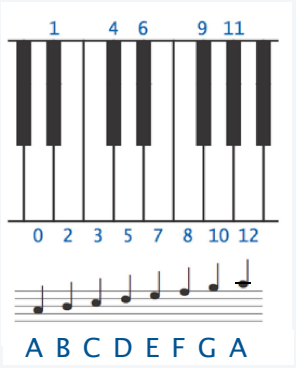
Play that tune

Read a list of tones and durations from standard input to play a tune.

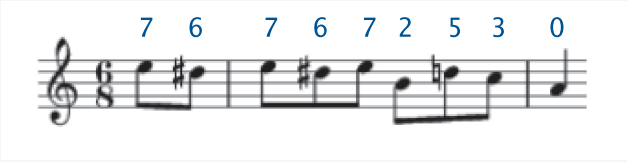
```
public class PlayThatTune
{
   public static void main(String[] args)
   {
      double tempo = Double.parseDouble(args[0]);
      while (!StdIn.isEmpty())
      {
         int pitch = StdIn.readInt();
         double duration = StdIn.readDouble() * tempo;
         double hz = 440 * Math.pow(2, pitch / 12.0);
         double[] a = PlayThatNote.tone(hz, duration);
         StdAudio.play(a);
      }
      StdAudio.close();
   }
}
```

control tempo from command line

```
% more < elise.txt
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25
...
```




```
% java PlayThatTune 2.0 < elise.txt
```




```
% java PlayThatTune 1.0 < elise.txt
```

Pop quiz 2 on functions

Q. What sound does the following program produce?

```
public class PQfunctions2
{
    public static void main(String[] args)
    {
        int N = (int) (44100 * 11);
        double[] a = new double[N+1];
        for (int i = 0; i <= N; i++)
            a[i] = Math.random();
        StdAudio.play(a);
    }
}
```

Pop quiz 2 on functions

Q. What sound does the following program produce?

```
public class PQfunctions2
{
    public static void main(String[] args)
    {
        int N = (int) (44100 * 11.0);
        double[] a = new double[N+1];
        for (int i = 0; i <= N; i++)
            a[i] = Math.random();
        StdAudio.play(a);
    }
}
```

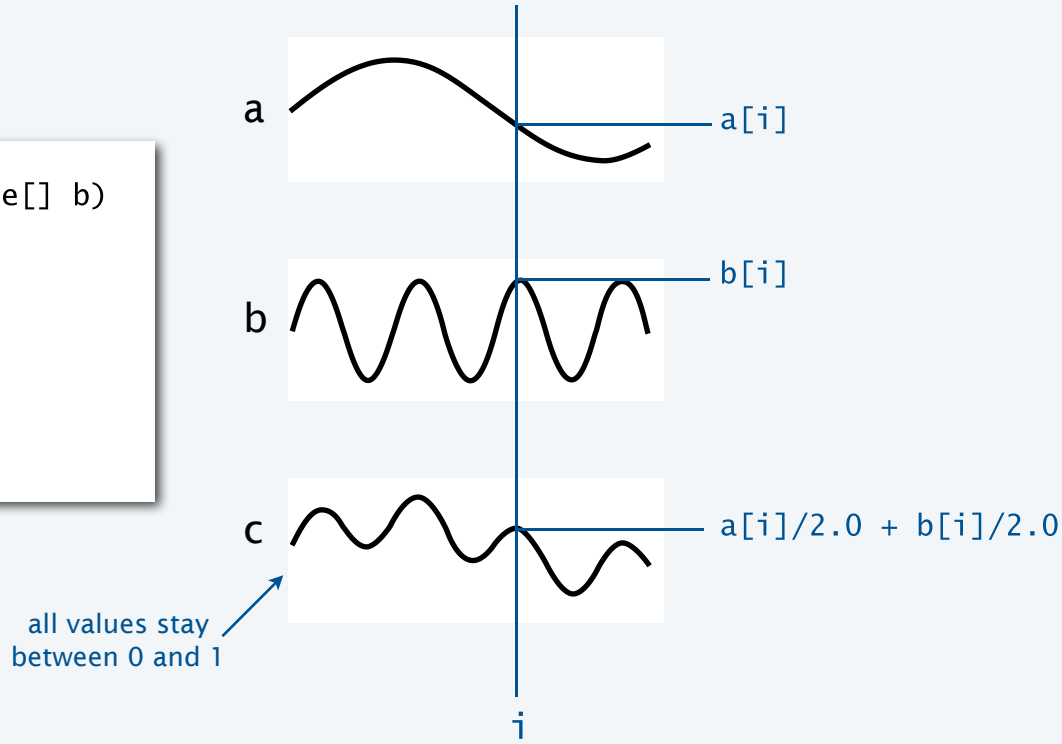
A. 11 seconds of pure *noise*.

```
% java PQfunctions2.java
```

Play that chord

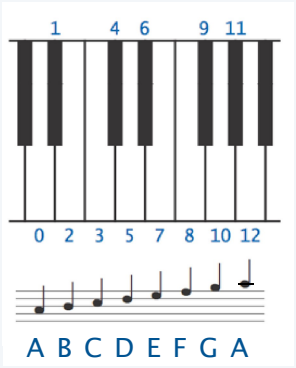
Produce chords by *averaging* waveforms.

```
public static double[] avg(double[] a, double[] b)
{
   double[] c = new double[a.length];
   for (int i = 0; i < a.length; i++)
      c[i] = a[i]/2.0 + b[i]/2.0;
   return c;
}
```



Play that chord implementation

```
public class PlayThatChord
{
    public static double[] avg(double[] a, double[] b)
    {  /* See previous slide. */ }

    public static double[] chord(int pitch1, int pitch2, double d)
    {
        double hz1 = 440.0 * Math.pow(2, pitch1 / 12.0);
        double hz2 = 440.0 * Math.pow(2, pitch2 / 12.0);
        double[] a = PlayThatNote.tone(hz1, d);
        double[] b = PlayThatNote.tone(hz2, d);
        return avg(a, b);
    }

    public static void main(String[] args)
    {
        int pitch1 = Integer.parseInt(args[0]);
        int pitch2 = Integer.parseInt(args[1]);
        double duration = Double.parseDouble(args[2]);
        double[] a = chord(pitch1, pitch2, duration);
        StdAudio.play(a);
    }
}
```




```
% java PlayThatChord 0 3 5.0
```

```
% java PlayThatChord 0 12 5.0
```

Play that tune (deluxe version)

Add harmonics to PlayThatTune to produce a more realistic sound.

Function to add harmonics to a tone

```
public static double[] note(int pitch, double duration)
{
      double hz = 440.0 * Math.pow(2, pitch / 12.0);
      double[] a  = tone(1.0 * hz, duration);
      double[] hi = tone(2.0 * hz, duration);
      double[] lo = tone(0.5 * hz, duration);
      double[] harmonic  = sum(hi, lo);
      return avg(a, harmonic);
}
```

```
% java PlayThatTune 1.5 < elise.txt
```

Program 2.1.4 in text (with tempo added)