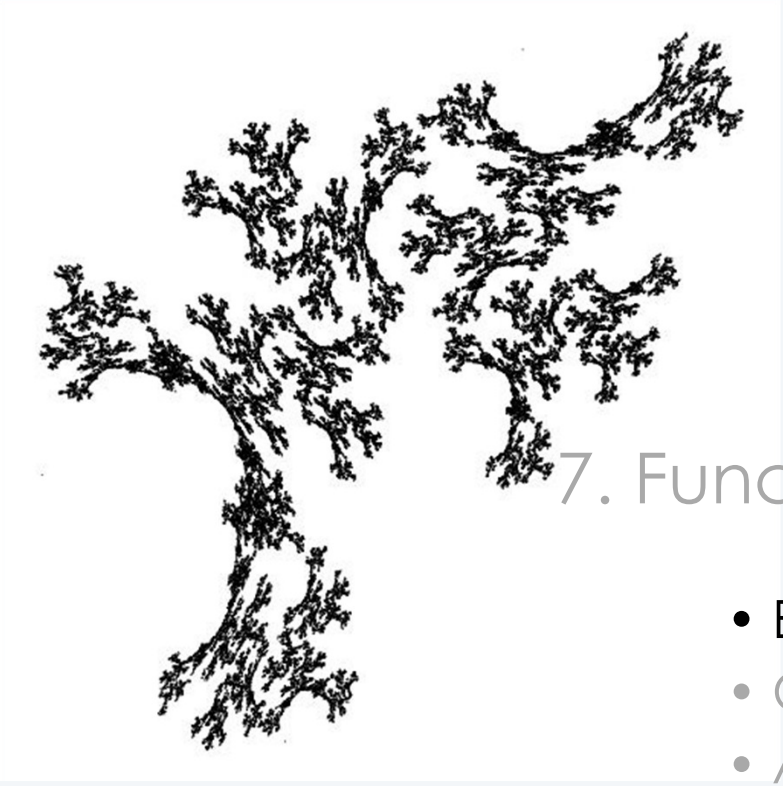
INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

7. Functions and Modules

http://introcs.cs.rutgers.edu



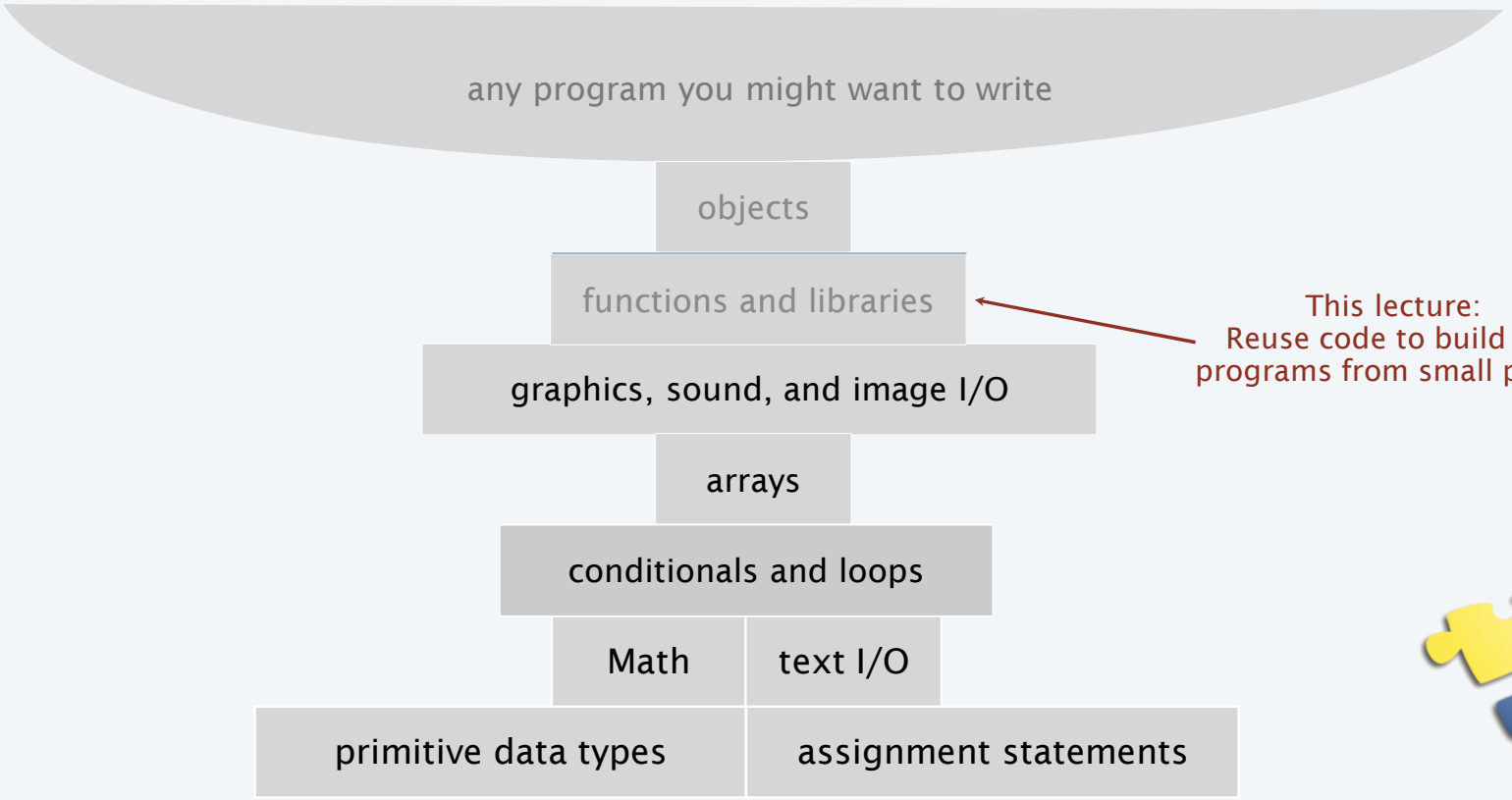
INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

7. Functions and Modules

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

Adopted and modified from Slides by Sedgewick and Wayne

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).

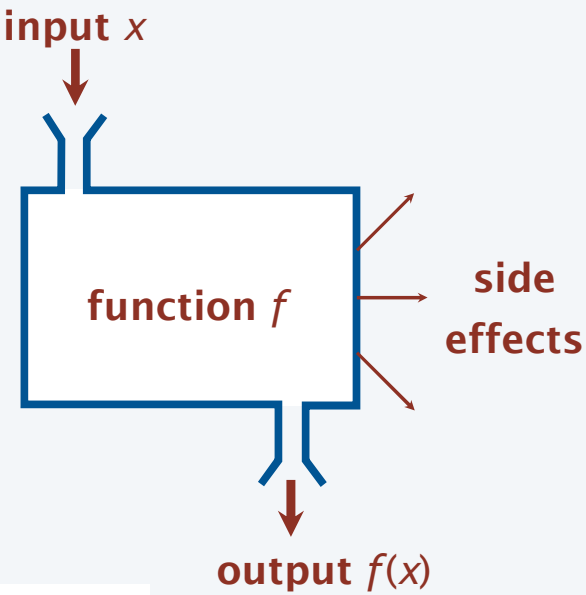
Functions (static methods)

Explain the meaning and use of *static methods* in Java.

LO 7.1a

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.

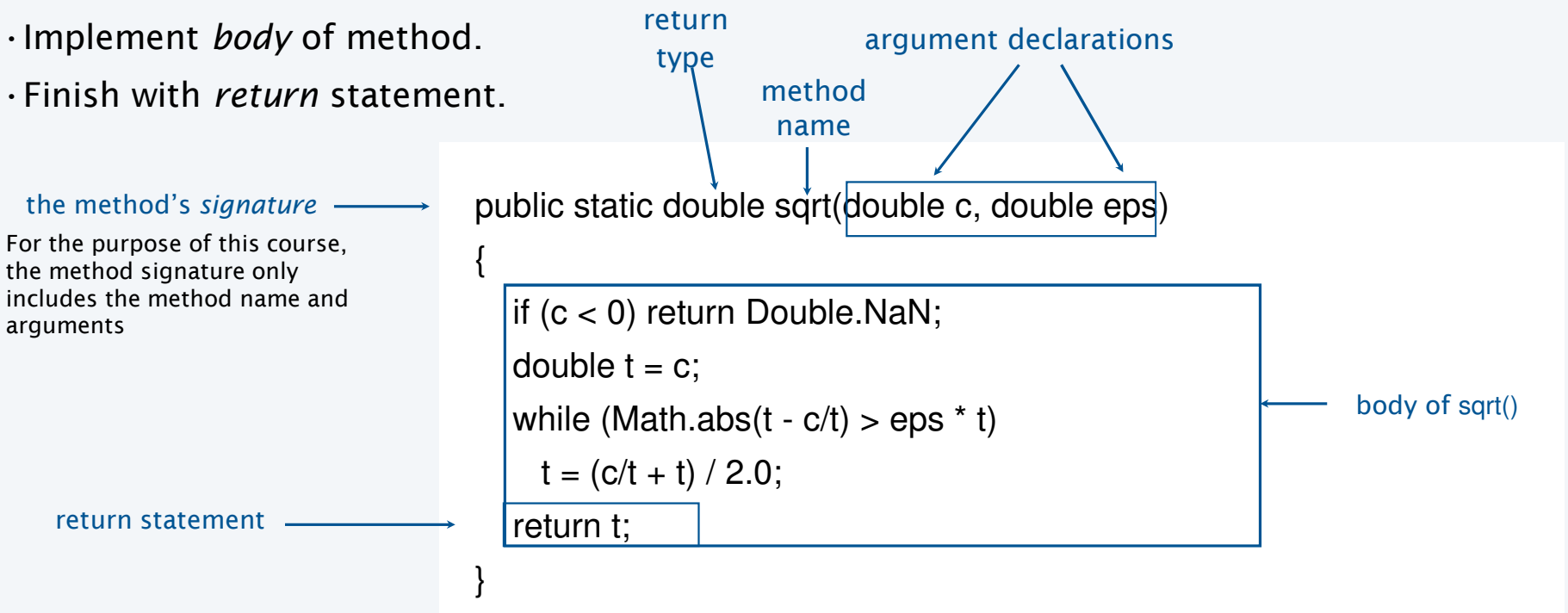
Examples seen so far

- 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/module

Use pre-existing functions/modules when writing program code.

LO 7.1b

A library is a set of functions.

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

module named Newton.java

sqrt() method

main() method

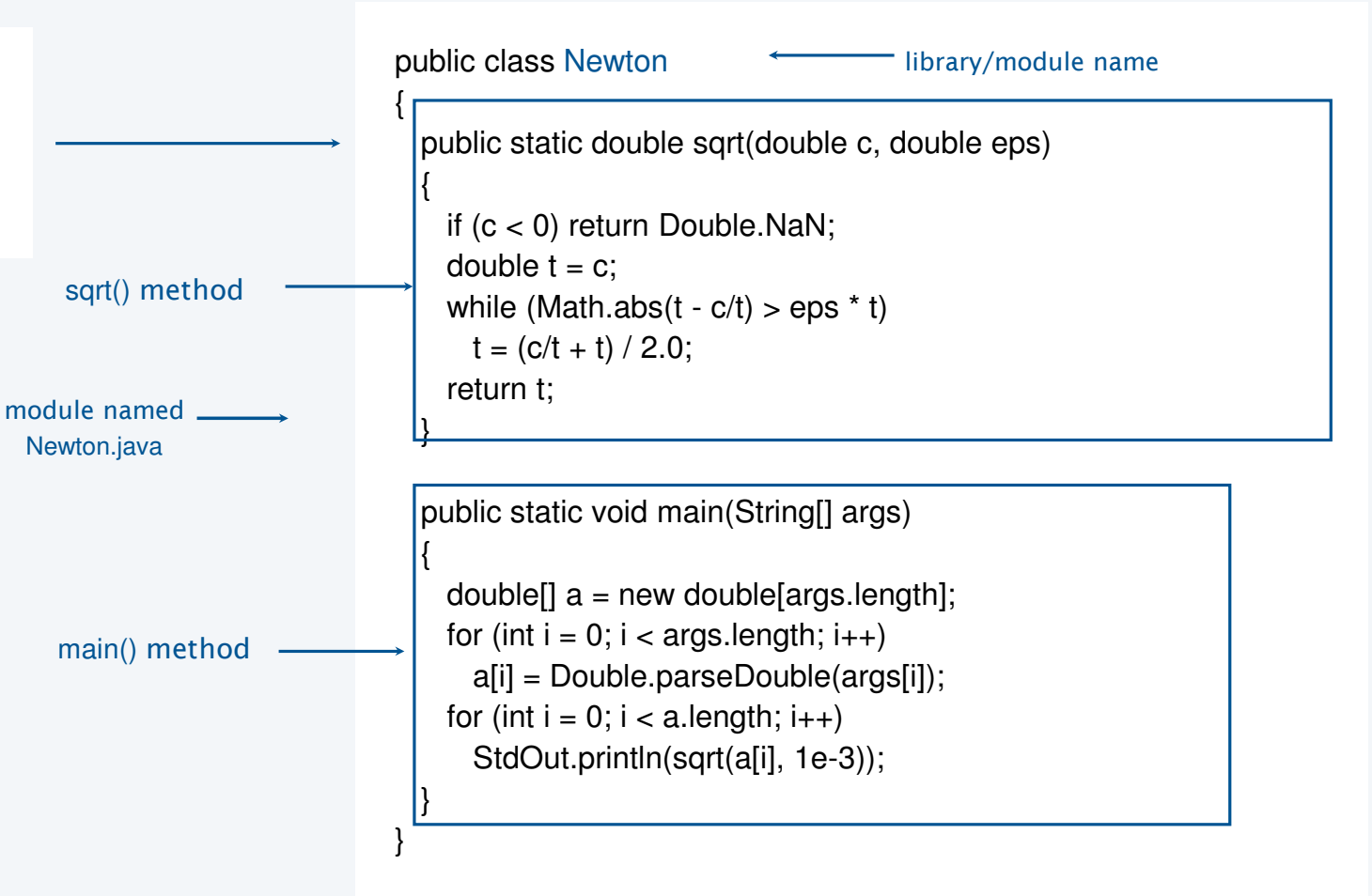
```
public class Newton        <-- library/module name
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   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));
   }
}
```

Static methods and parameters

(7.1c) Define and use static methods with and without parameters in program code.
(7.1d) Define and use static methods with and without return values in program code.

LO 7.1cd

Key point. Static methods can pass parameters to calling function

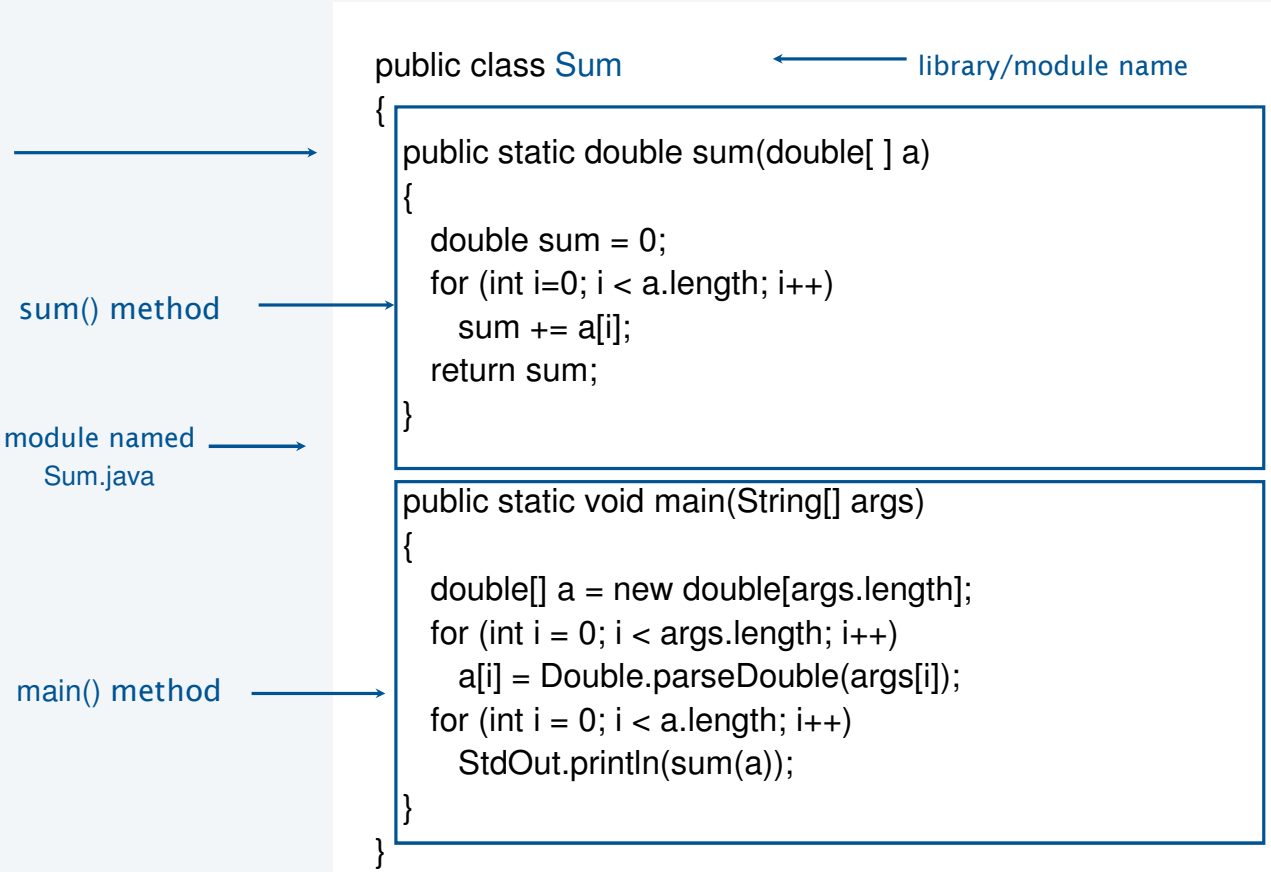


Static methods with array parameters

Define and use static methods that include arrays as parameters or return types.

LO 7.1e

Key point. Static methods can receive array parameters

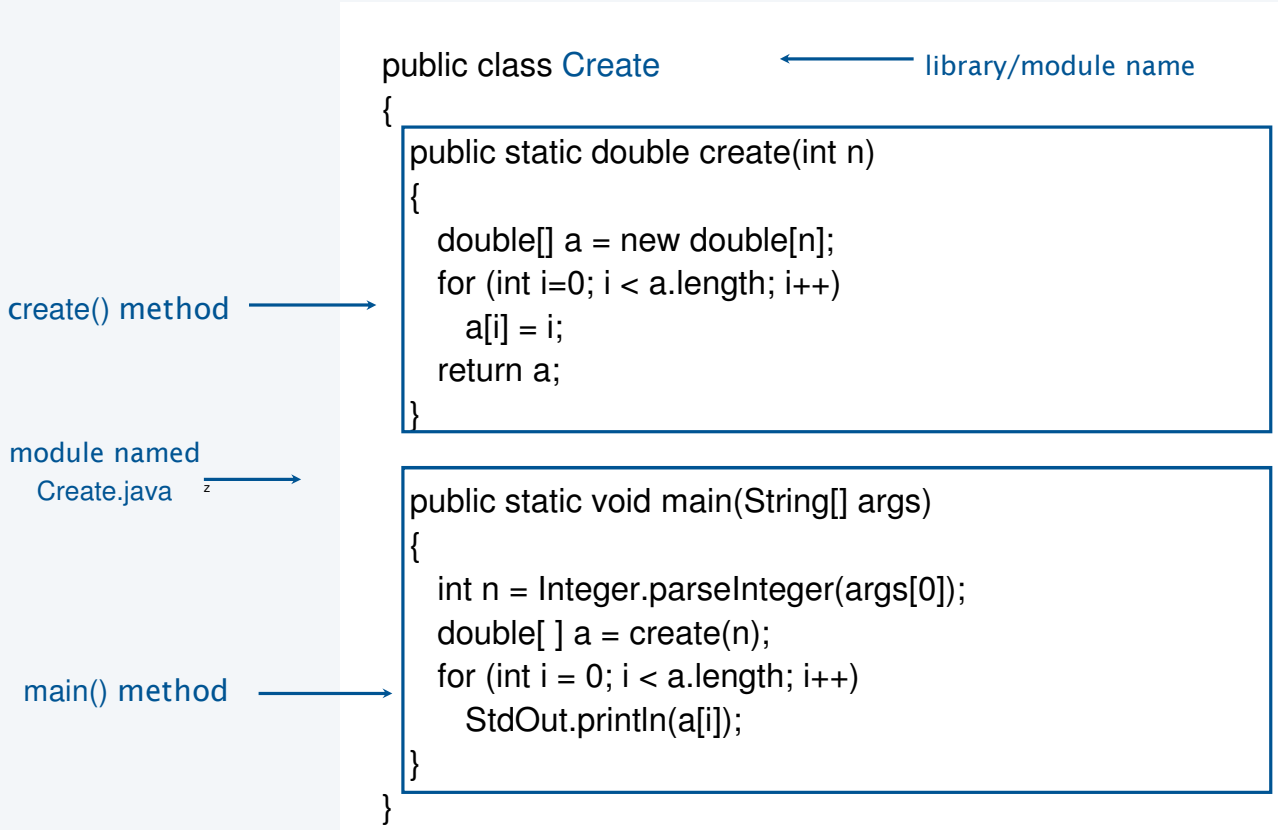


Static methods with array return parameters

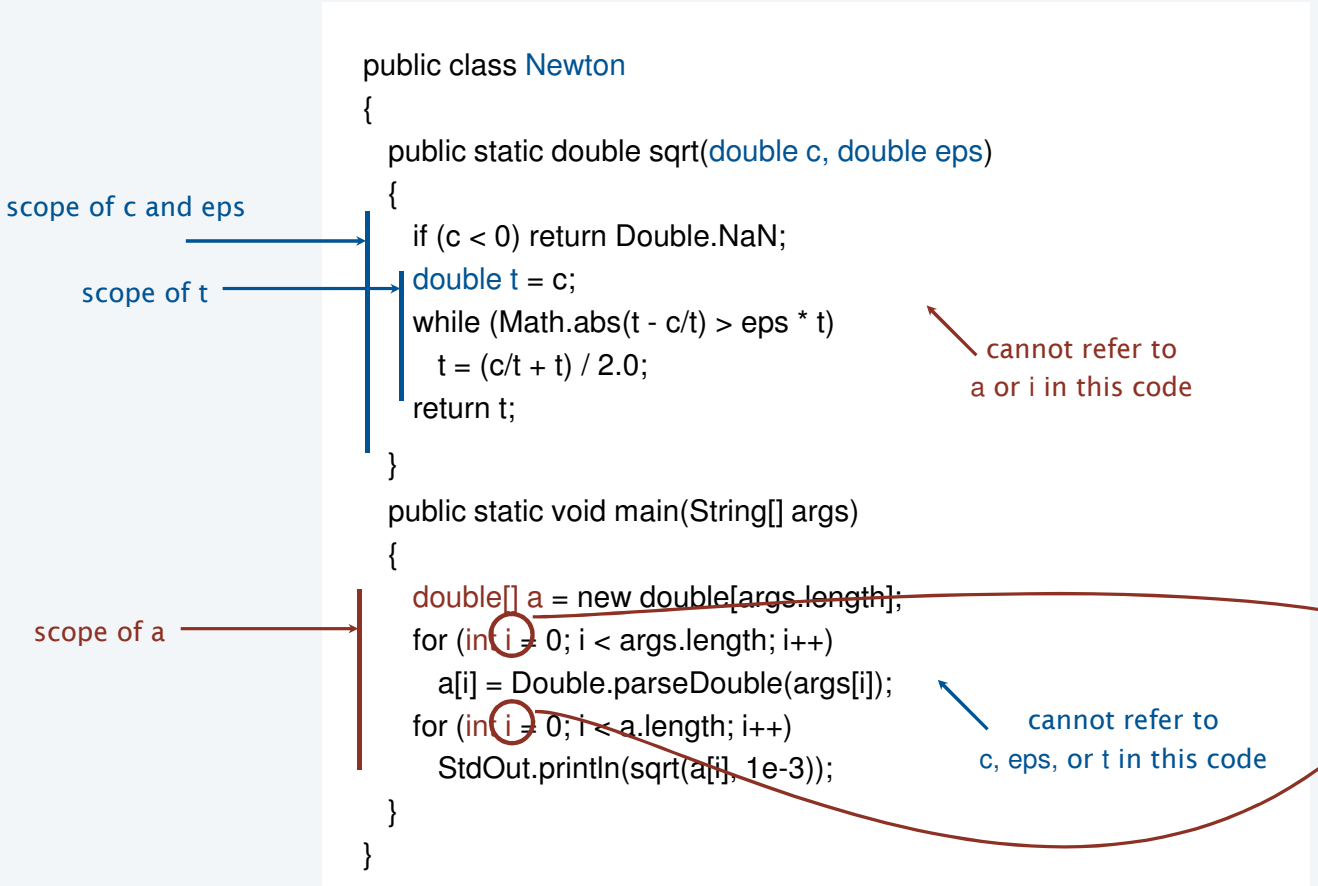
Define and use static methods that include arrays as parameters or return types.

LO 7.1e

Key point. Static methods can return an array



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

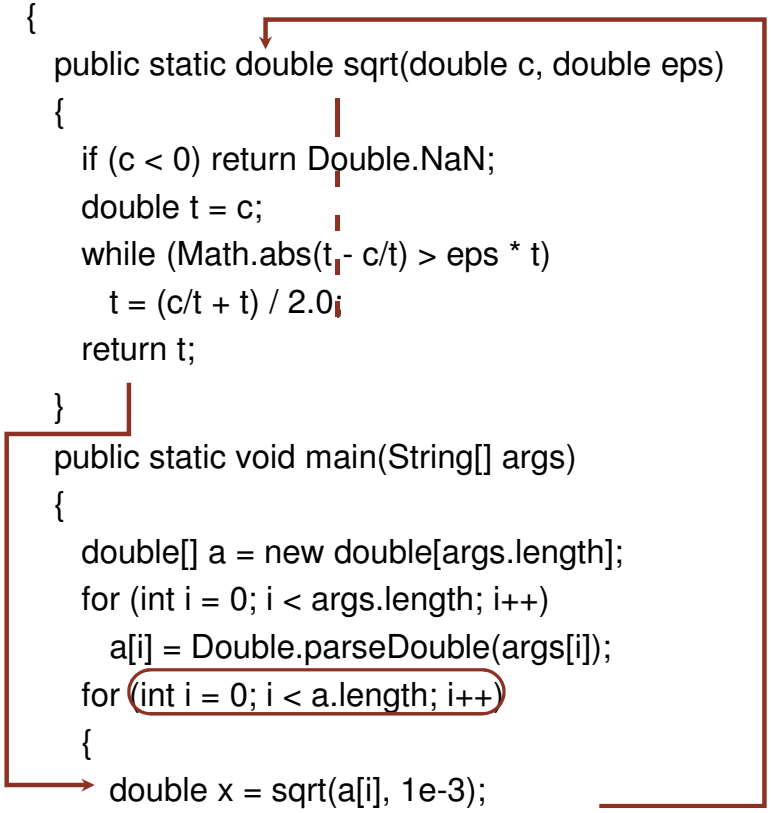


In a Java library, a variable's scope is the code following its declaration, in the same block.

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
1.0	1.0

i	a[i]
0	1.0
1	2.0
2	3.0

c	t
2.0	2.0
	1.5
	1.417
	1.414

c	t
3.0	3.0
	2.0
	1.75
	1.732

3

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

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

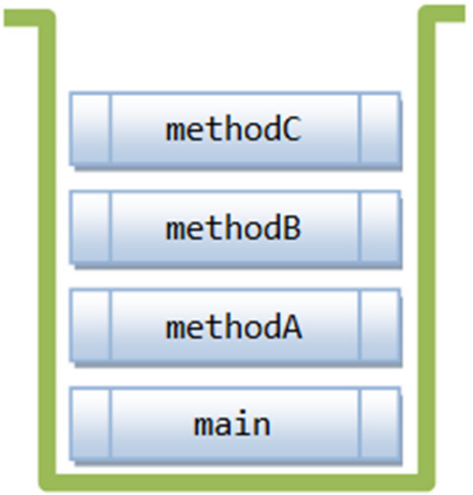
Function call stack

Explain and illustrate the call stack for a program that includes multiple method calls.

LO 7.1f

Q. What happens when you call a function?

```
public class Demo
{
   public static int methodA()  { method(); // calls method B}
   public static int methodB()  { methodC(); // calsl method C}
    public static int methodC() { …}

  public static void main(String[] args)
   {
      // call methodA
   }
}
```



Method Call Stack
(Last-in-First-out Queue)

A. A function call stack is created. A call stack contains information about function variables and other system information. In the above case, methodA call from main result in series of function calls.

Tracing a function call stack

Trace and write programs that include methods having multiple return statements. LO 7.1g

Q. What is the output of the following code?

```
public class Demo
{
   public static int add(int a, int b)  { return a+b; }
   public static int multiply(int a, int b)  { return a*b;}
   public static int square(int a) { return a*a;}

  public static void main(String[] args)
   {
      int x = 3, y = 4;
      int z = add(multiply(square(x), y), add(x, square(y));
   }
}
```

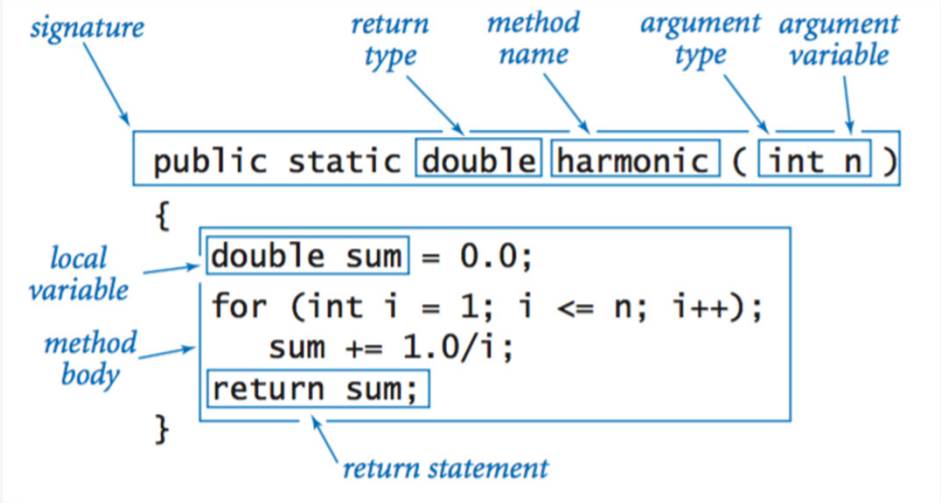
A. This returns the value 55 (explain)

Understanding the method signature

Describe the meaning of each part of a method signature.

LO 7.1i

A method signature defines the form of the method.



Static methods allow method to be called without an object (We will study objects later).

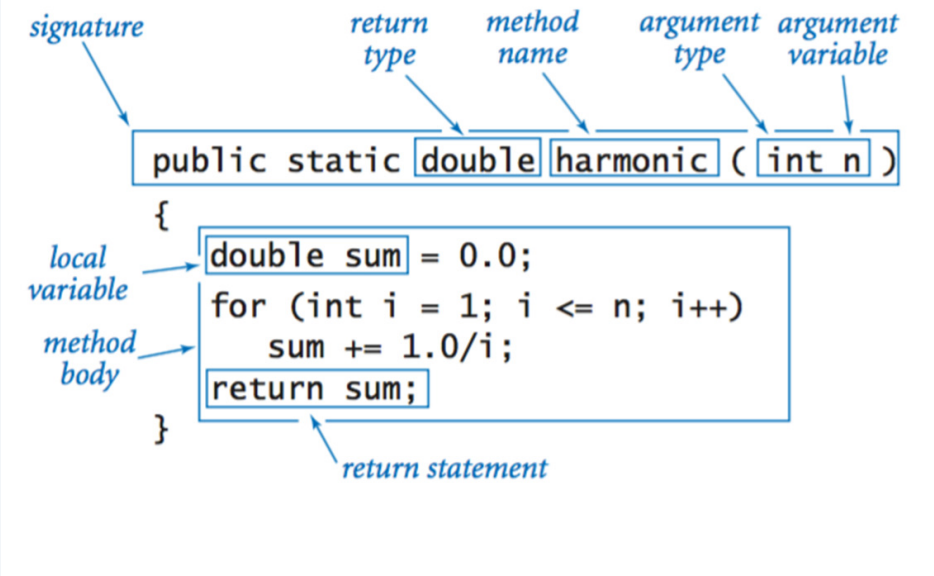
For the purpose of this course, the method signature only includes the method **name** and **arguments**

Method local variables and parameter variables

Explain the difference between local variables and parameter variables.

LO 7.1k

The scope of a variable depends on how it is defined.



n is a parameter variable The scope of n is the entire function body

sum is a local variable The scope of sum is also the entire function body

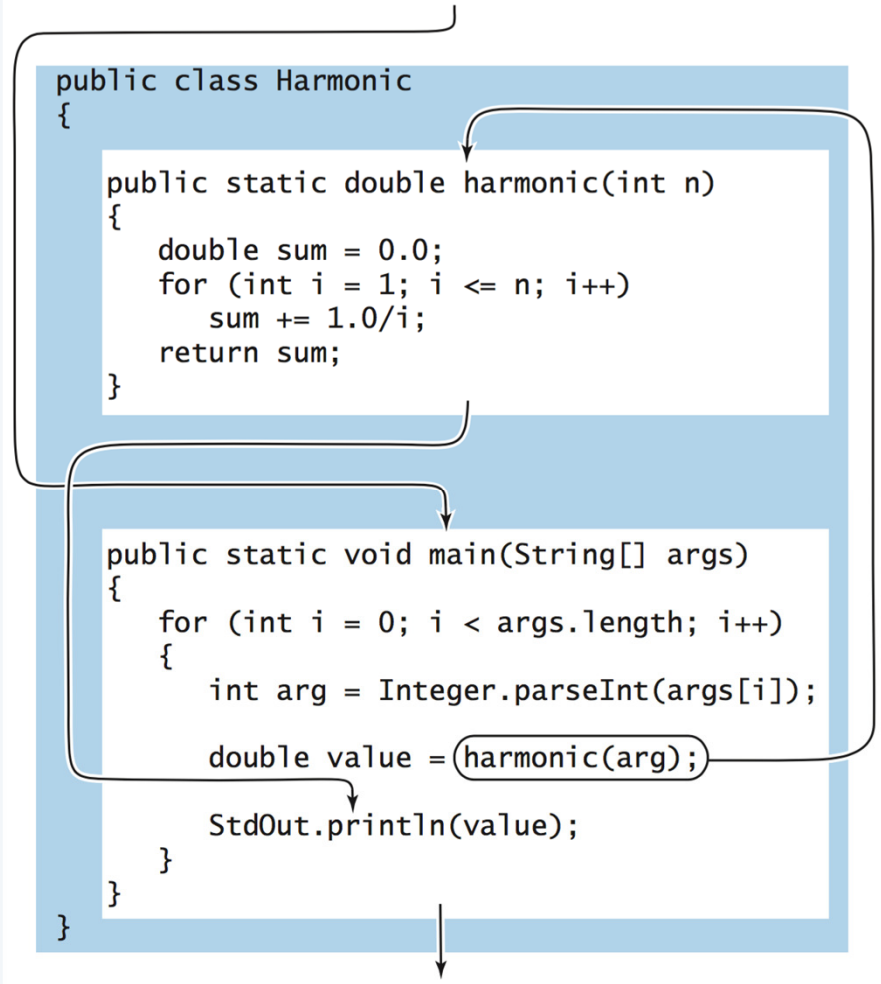
Loop control i is a local variable whose scope is the loop body

For the purpose of this course, the method signature only includes the method name and arguments

Method implementation and method call

Explain the difference between a method implementation and a method call.

LO 7.1I



```
public class Harmonic
{
   public static double harmonic(int n)
   {
      double sum = 0.0;
      for (int i = 1; i <= n; i++)
         sum += 1.0/i;
      return sum;
   }

   public static void main(String[] args)
   {
      for (int i = 0; i < args.length; i++)
      {
         int arg = Integer.parseInt(args[i]);

         double value = harmonic(arg);

         StdOut.println(value);
      }
   }
}
```

The method implementation is how the method is defined.

The method call is how the method is used.

Overloaded methods

Trace and write programs involving overloaded methods.

LO 7.1m

```
/******************************************************************************
 *  Compilation:  javac Overloaded.java
 *  Execution:    java Overloaded
 *
 ******************************************************************************/

public class Overloaded {

    public static void f(int x, double y) {
        StdOut.println("f(int, double)");
    }

    public static void f(double x, int y) {
        StdOut.println("f(double, int)");
    }

    public static void f(double x, double y) {
        StdOut.println("f(double, double)");
    }

    public static void main(String[] args) {
        f(1.0, 17);
        f(1, 17.0);
        f(1.0, 17.0);
        //// f(1, 17);        // compile-time error: reference to f is ambiguous
    }
```

Same method name f, but different argument types.

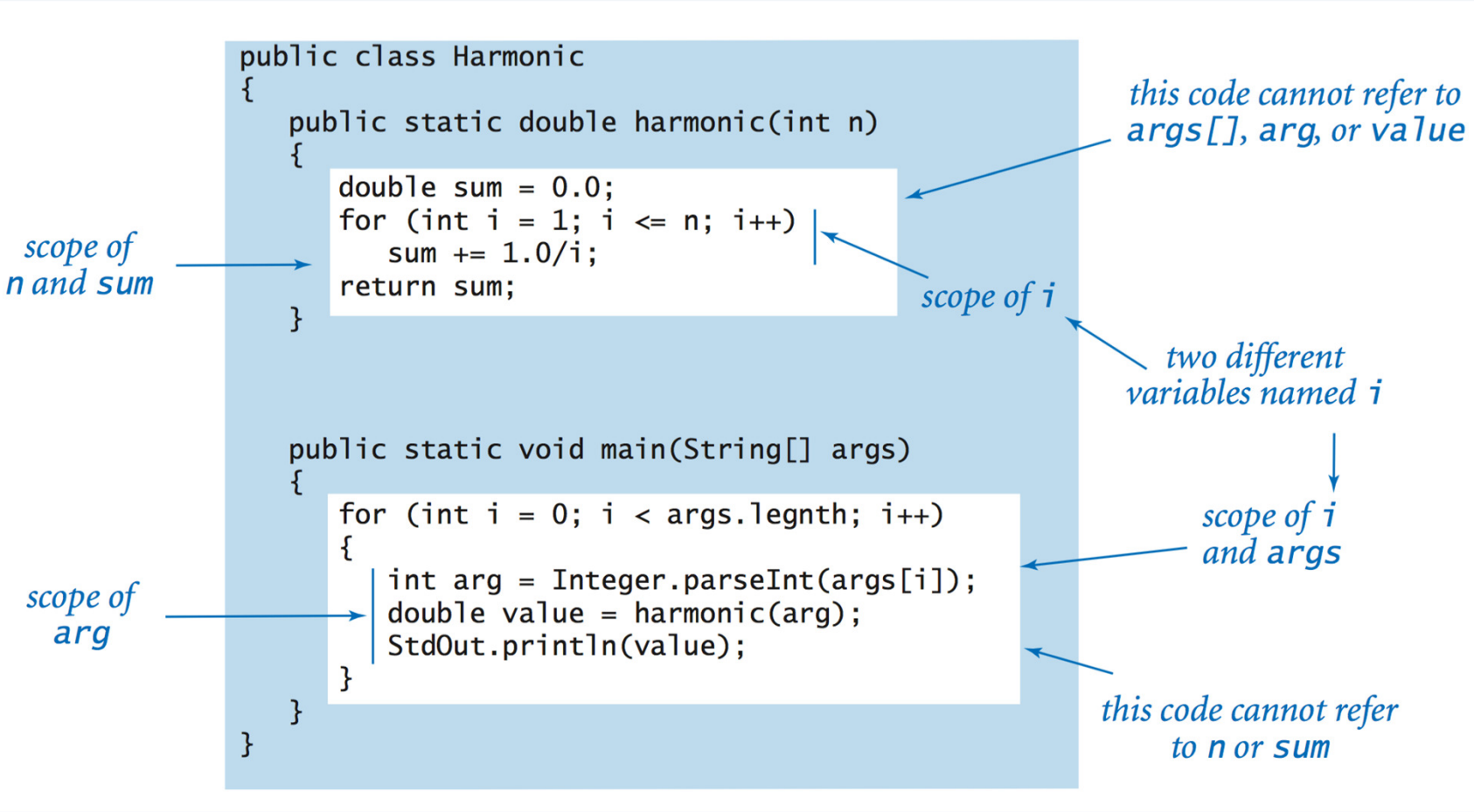
Compiler will decide which method to use based on argument type

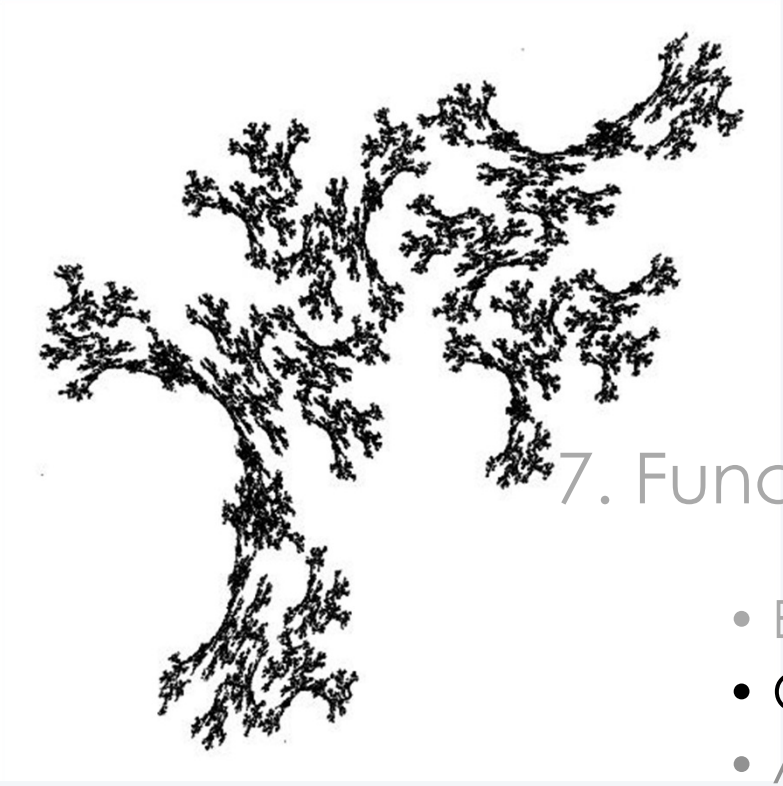
Source: introcs.cs.princeton.edu

Scope of a variable

Identify the scope of a variables in a program with multiple methods and method calls.

LO 7.1n





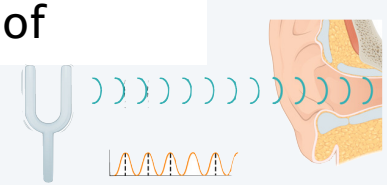
INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

7. Functions and Modules

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

Sound

Sound is the perception of the vibration of molecules.

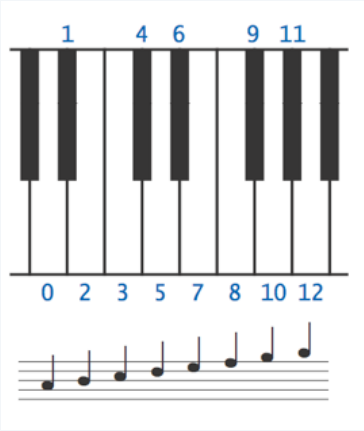


A musical tone is a steady periodic sound.

A pure tone is a sinusoidal waveform.

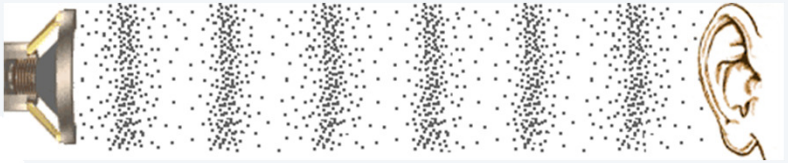
Western musical scale

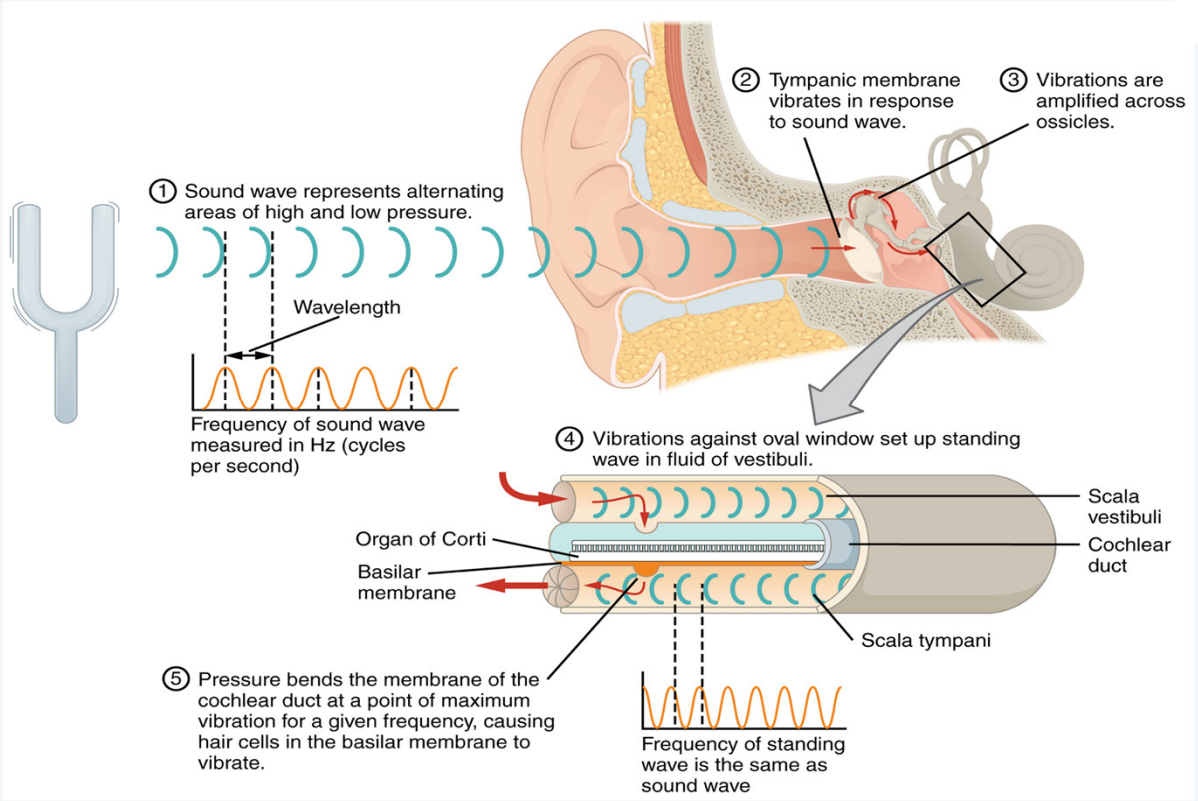
- Concert A is 440 Hz.
- 12 notes, logarithmic scale.

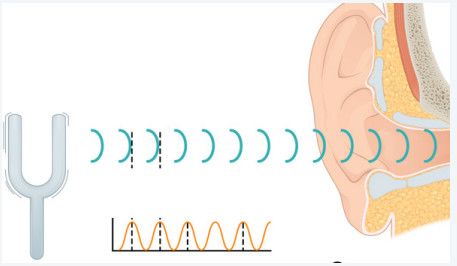


pitch	*i*	frequency ($440*2^{i/12}$)	sinusodial waveform
A	0	440	
A♯ / B♭	1	466.16	
B	2	493.88	
C	3	523.25	
C♯ / D♭	4	554.37	
D	5	587.33	
D♯ / E♭	6	622.25	
E	7	659.26	
F	8	698.46	
F♯ / G♭	9	739.99	
G	10	783.99	
G♯ / A♭	11	830.61	
A	12	880	

Crash course in sound







http://commons.wikimedia.org/wiki/File:1405_Sound_Waves_and_the_Ear.jpg

Digital audio

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

same as when plotting a function

1/40 second of concert A

samples/sec	*samples*	sampled waveform
5,512	137	
11,025	275	
22,050	551	
44,100	1102	

CD standard →

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

StdAudio library

Developed for the textbook, 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.

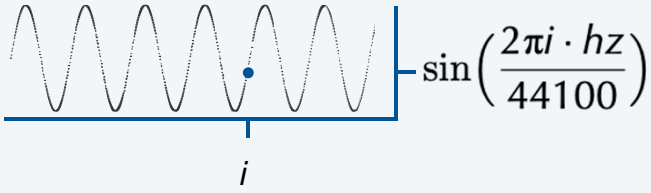
public class StdAudio		
void play(String file)		*play the given* .wav *file*
void play(double[] a)		*play the given sound wave*
void play(double x)	**API**	*play the sample for 1/44100 second*
void save(String file, double[] a)		*save to a* .wav *file*
double[] read(String file)		*read from a* .wav *file*

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
```

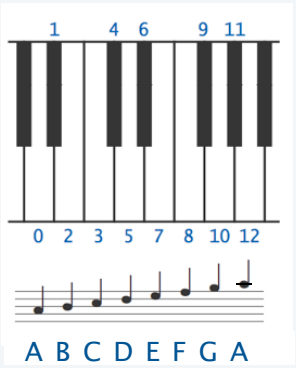
Play that tune

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

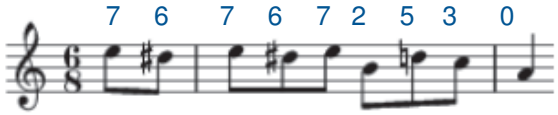
```
public class PlayThatTune
{
   public static void main(String[] args)
   {
      double tempo = Double.parseDouble(args[0]);
      while (!StdIn.isEmpty())
      {
         int pitch = StdIn.readInt();
         double duration = StdIn.readDouble() * tempo;
         double hz = 440 * Math.pow(2, pitch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

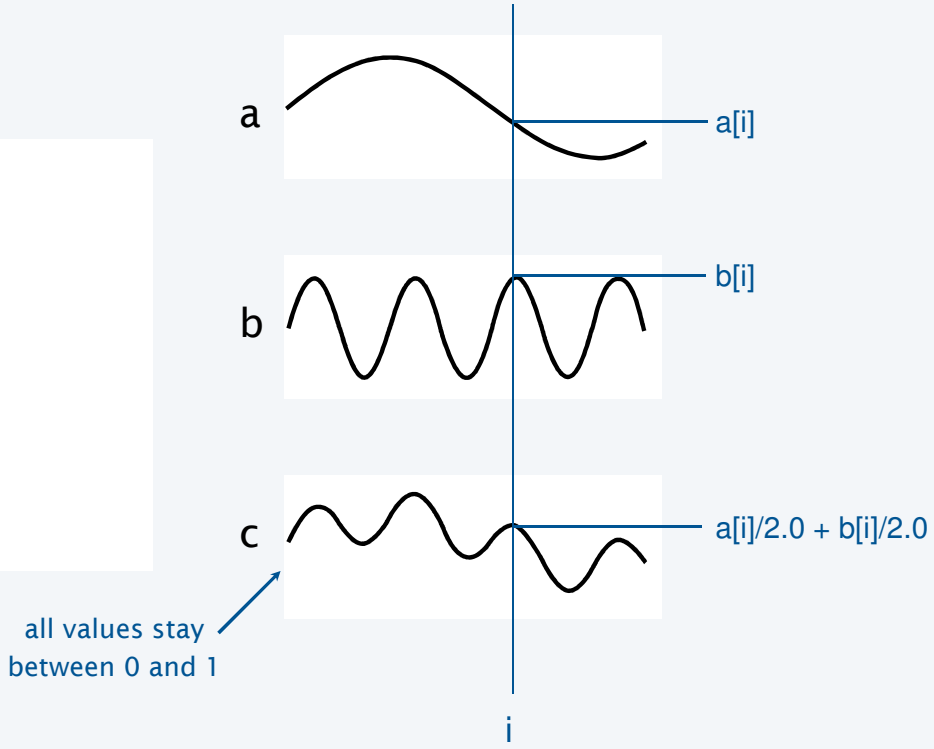
A. 11 seconds of pure *noise*.

```
% java PQfunctions2.java
```

Play that chord

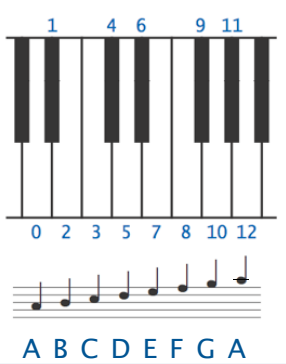
Produce chords by *averaging* waveforms.

```
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)

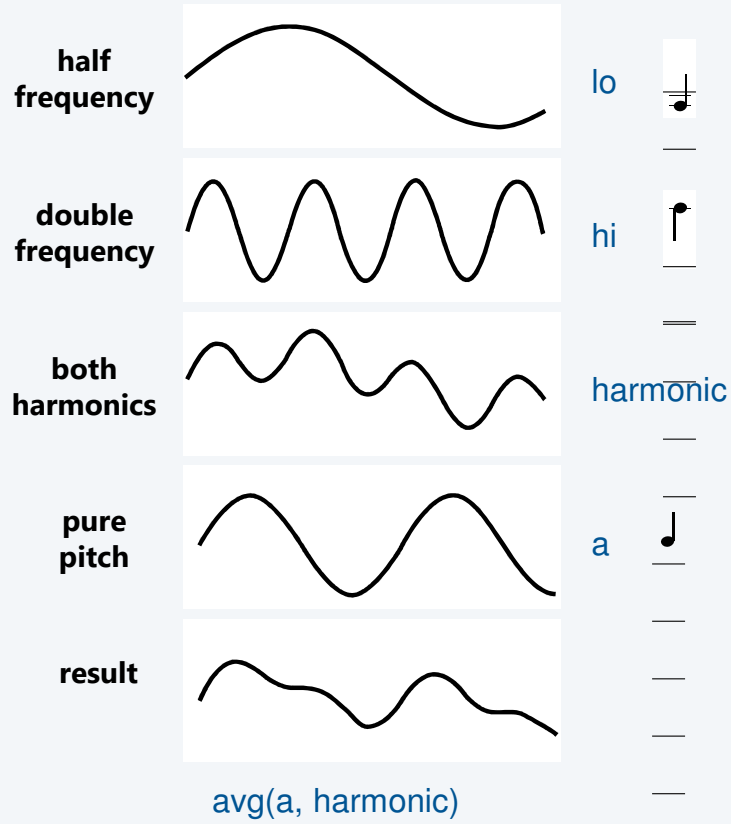
Add harmonics to PlayThatTune to produce a more realistic sound.

Function to add harmonics to a tone

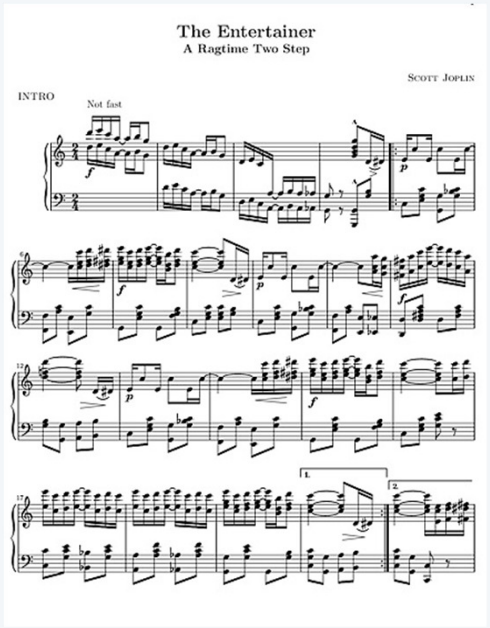
```
public static double[] note(int pitch, double duration)
{
   double hz = 440.0 * Math.pow(2, pitch / 12.0);
   double[] a  = tone(1.0 * hz, duration);
   double[] hi = tone(2.0 * hz, duration);
   double[] lo = tone(0.5 * hz, duration);
   double[] harmonic  = sum(hi, lo);
   return avg(a, harmonic);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

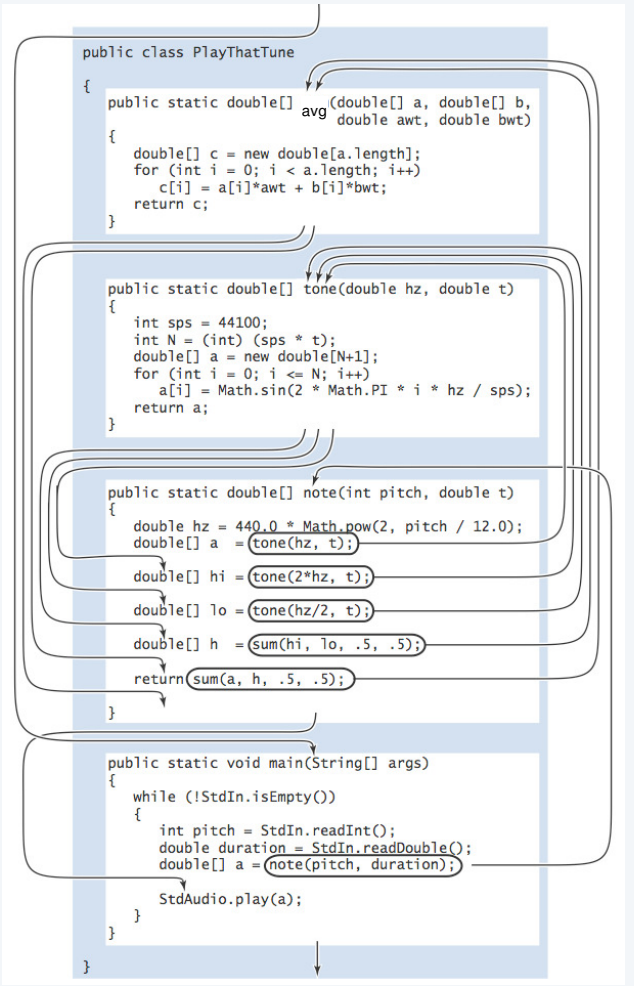
```
% java PlayThatTune 1.5 < elise.txt
```

Program 2.1.4 in text (with tempo added)



Digital audio (summary)



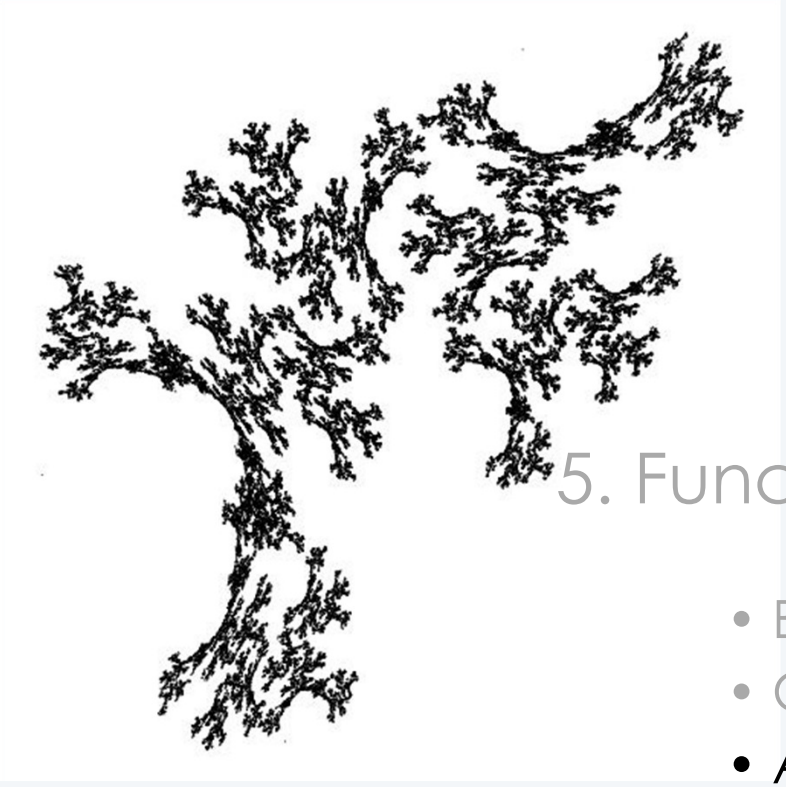



```
% java PlayThatTune 1.5 < entertainer.txt
```

Bottom line. *You* can write programs to manipulate sound.

This lecture: Case study of the utility of functions.

Upcoming assignment: Fun with musical tones.



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

5. Functions and Libraries

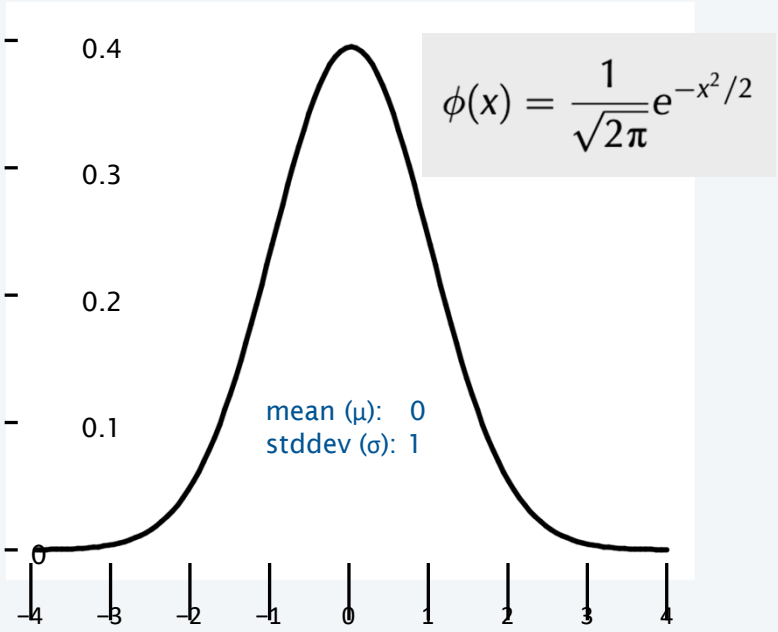
- Basic concepts
- Case study: Digital audio (optional)
- Application: Gaussian distribution (optional)
- Modular programming

Gaussian distribution

Gaussian distribution.

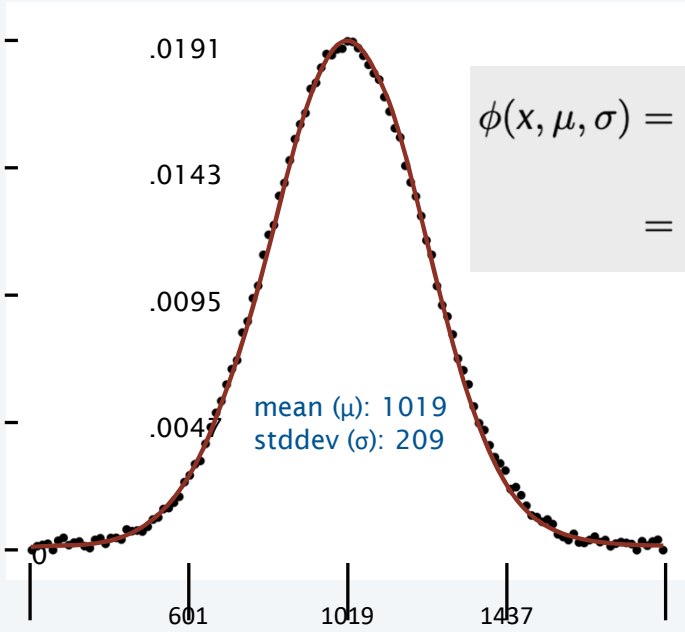
- A mathematical model used successfully for centuries.
- "Bell curve" fits experimental observations in many contexts.

Gaussian probability density function (pdf)



$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

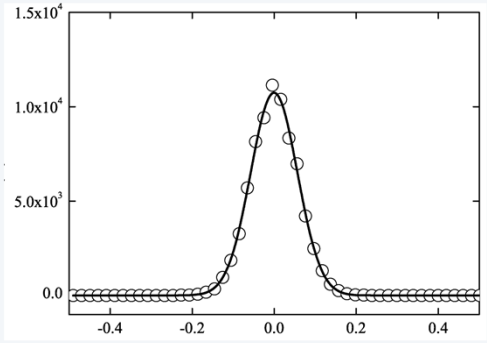
Example: SAT scores in 20xx (verbal + math)



$$\phi(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$$
$$= \frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right)$$

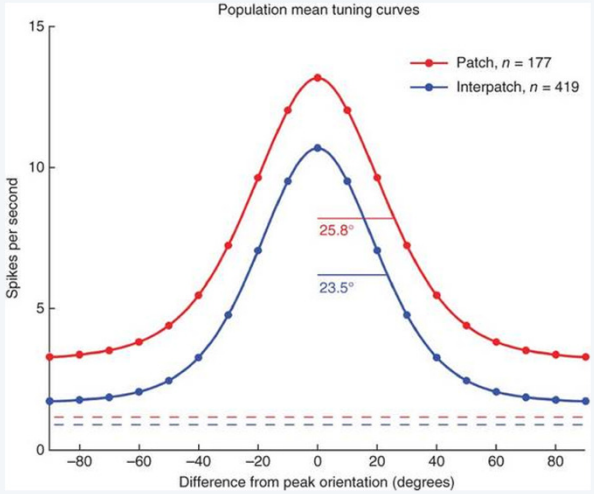
Gaussian distribution in the wild

Polystyrene particles in glycerol

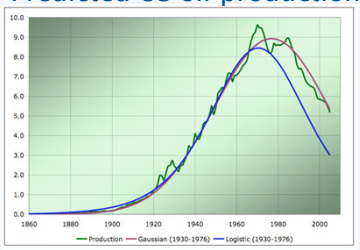


German money

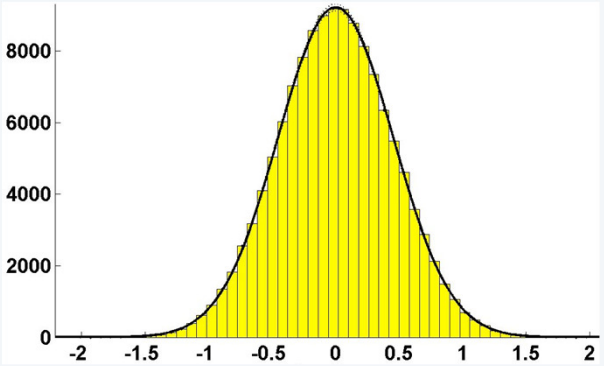
Cytochrome oxidase patches
in macaque primary visual cortex



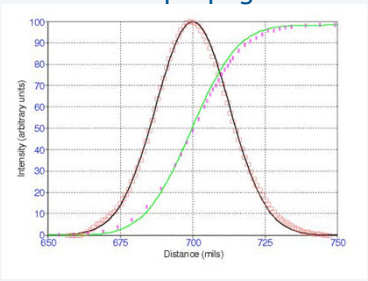
Predicted US oil production



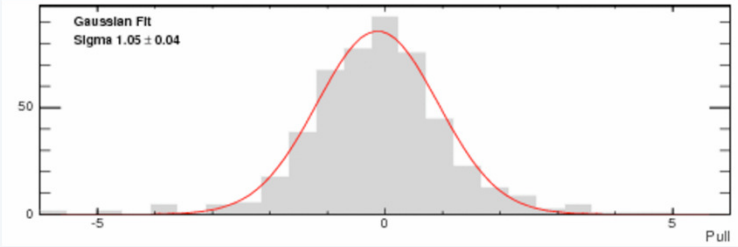
Calibration of optical tweezers



Laser beam propagation

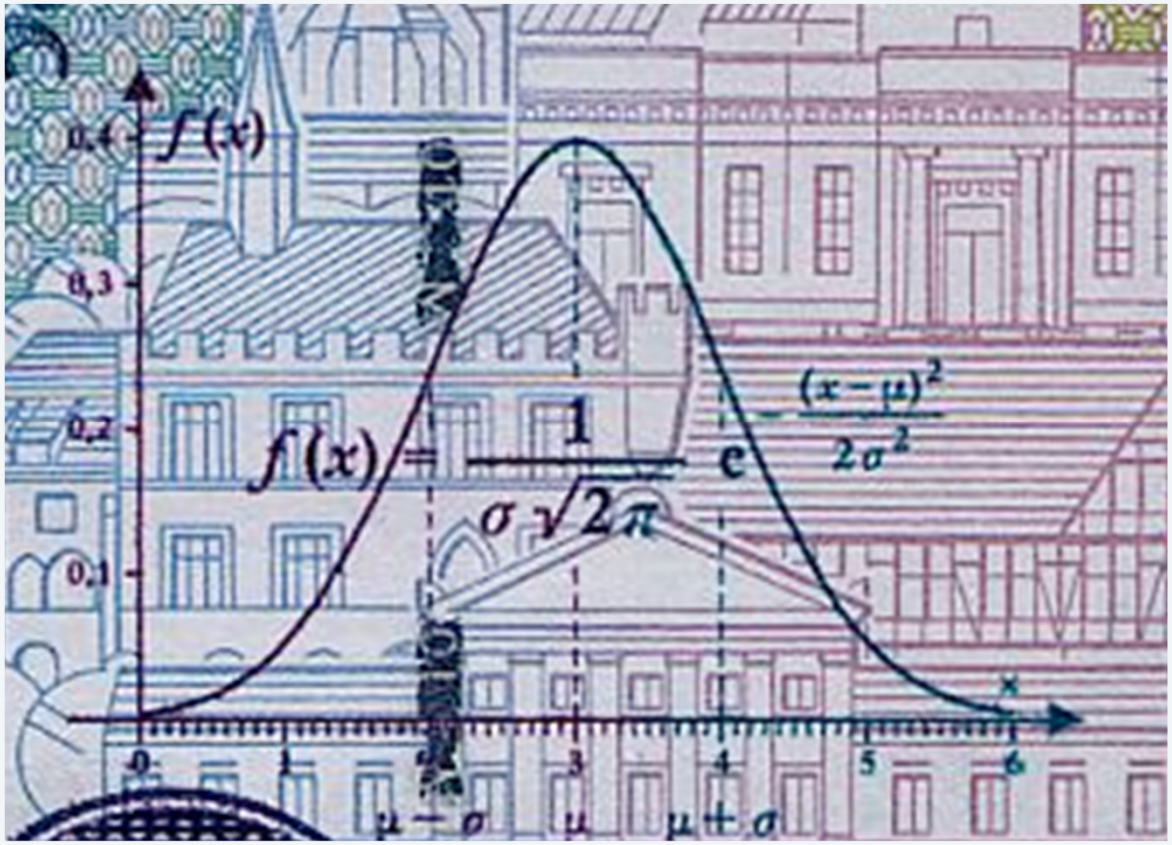


Polarized W bosons from top-quark decay



German 10 mark note (1993; discontinued)

German money off by a factor of sigma?



Defining a library of functions

Q. Is the Gaussian pdf Φ implemented in Java's Math library?

A. No.

Q. Why not?

A. Maybe because it is so easy for you to do it yourself.

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$\phi(x, \mu, \sigma) = \frac{1}{\sigma}\phi\left(\frac{x - \mu}{\sigma}\right)$$

```
public class Gaussian
{
   public static double pdf(double x)
   {
      return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI);
   }

   public static double pdf(double x, double mu, double sigma)
   {
      return pdf((x - mu) / sigma) / sigma;
   }

   // Stay tuned for more functions.
}
```

call a function in another module

module named Gaussian.java

functions with different signatures are different (even if names match)

Functions and libraries provide an easy way for any user to *extend* the Java system.

Gaussian cumulative distribution function

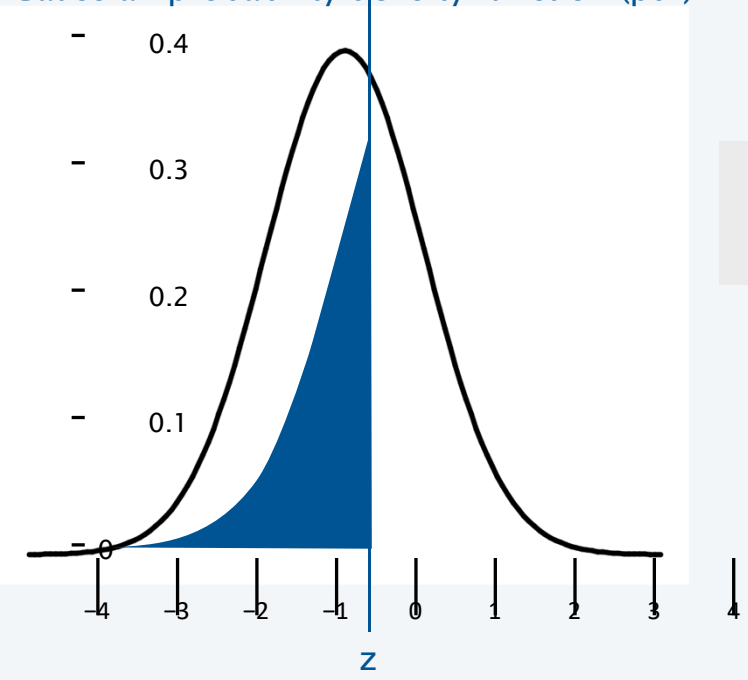
Q. What percentage of the total is less than or equal to *z*?

Q. (equivalent). What is the area under the curve to the left of *z*?

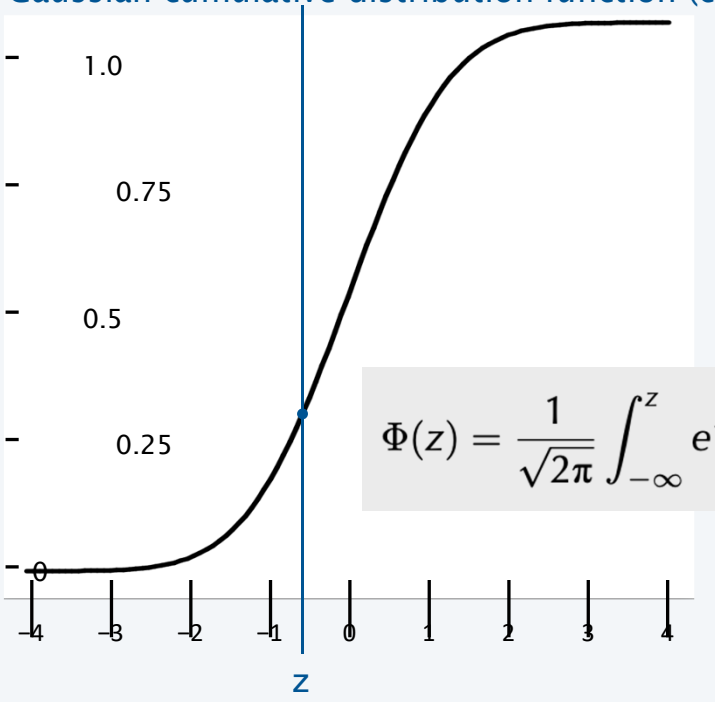
A. Gaussian *cumulative distribution function*.

$$\Phi(x, \mu, \sigma) = \Phi\left(\frac{x - \mu}{\sigma}\right)$$

Gaussian probability density function (pdf)



Gaussian cumulative distribution function (cdf)

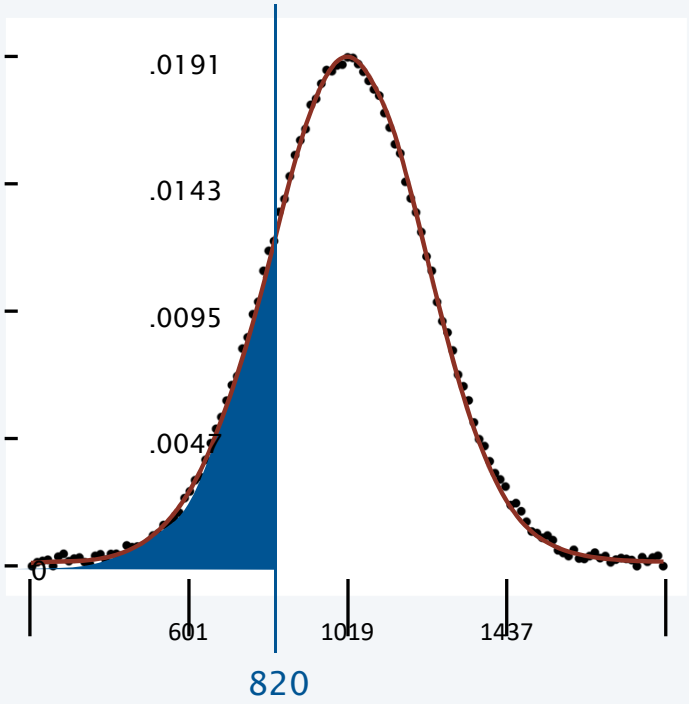


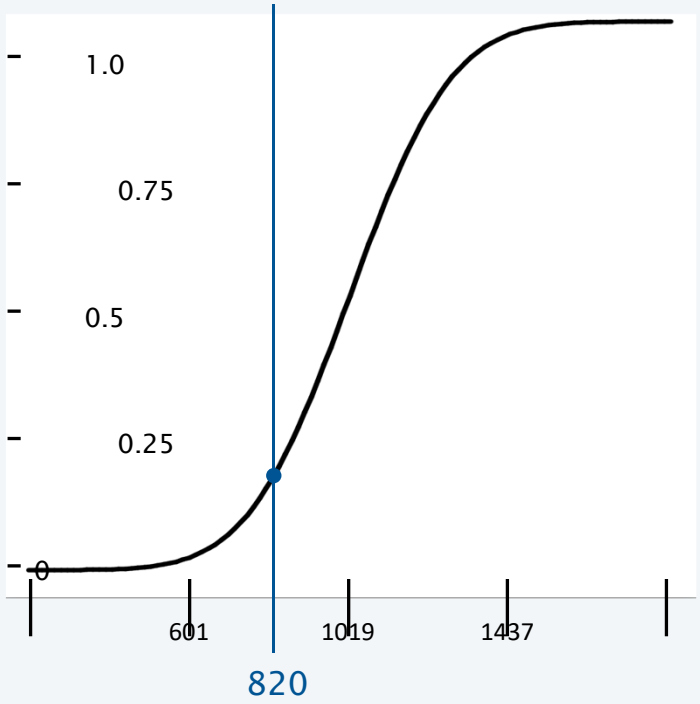
$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-x^2/2} dx$$

Typical application: SAT scores

Q. In 20xx NCAA required at least 820 for Division I athletes.
What fraction of test takers did not qualify??

A. About 17%, since $\Phi(820, 1019, 209) = 0.17050966869132111\ldots$





Gaussian CDF implementation

Q. No closed form for Gaussian CDF Φ. How to implement?

A. Use Taylor series. $\Phi(z) \equiv \int_{-\infty}^{z} \phi(x)dx = \frac{1}{2} + \phi(z)\left(z + \frac{z^3}{3} + \frac{z^5}{3 \cdot 5} + \frac{z^7}{3 \cdot 5 \cdot 7} + \ldots\right)$

```
public static double cdf(double z)
{
   if (z < -8.0) return 0.0;
   if (z >  8.0) return 1.0;
   double sum = 0.0, term = z;
   for (int i = 3; sum + term != sum; i += 2)
   {
      sum  = sum + term;
      term = term * z * z / i;
   }
   return 0.5 + sum * pdf(z);
}
public static double cdf(double z, double mu, double sigma)
{  return cdf((z - mu) / sigma);  }
```

⟵ accurate to 15 places

Bottom line. 1,000 years of mathematical formulas at your fingertips.

Summary: a library for Gaussian distribution functions

Best practice

- Test all code at least once in main().
- Also have it do something useful.

Q. What fraction of SAT test takers did not qualify for NCAA participation in 20xx?

```
% java Gaussian 820 1019 209
0.17050966869132111
```

Fun fact

We use cdf() to evaluate randomness in submitted programs.

Bottom line

YOU can build a layer of abstraction to use in any future program.

```
public class Gaussian
{
  public static double pdf(double x)
  { return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI); }
  public static double pdf(double x, double mu, double sigma)
  { return pdf((x - mu) / sigma) / sigma; }
  public static double cdf(double z)
  {
    if (z < -8.0) return 0.0;
    if (z >  8.0) return 1.0;
    double sum = 0.0, term = z;
    for (int i = 3; sum + term != sum; i += 2)
    {
      sum  = sum + term;
      term = term * z * z / i;
    }
    return 0.5 + sum * pdf(z);
  }
  public static double cdf(double z, double mu, double sigma)
  { return cdf((z - mu) / sigma); }
  public static void main(String[] args)
  {
    double z = Double.parseDouble(args[0]);
    double mu = Double.parseDouble(args[1]);
    double sigma = Double.parseDouble(args[2]);
    StdOut.println(cdf(z, mu, sigma));
  }
```

Using a library

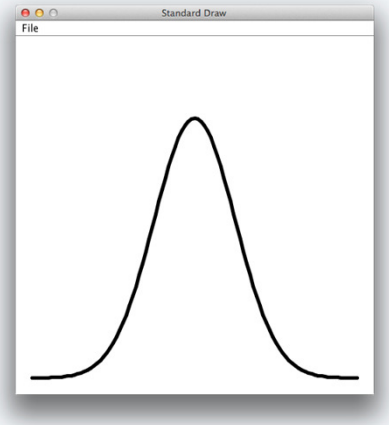
To use these methods in another program

Learn to use your OS "classpath" mechanism if you find yourself with too many copies.

- Put a copy of Gaussian.java in your working directory.
- Call Gaussian.pdf() or Gaussian.cdf() from any other module in that directory.

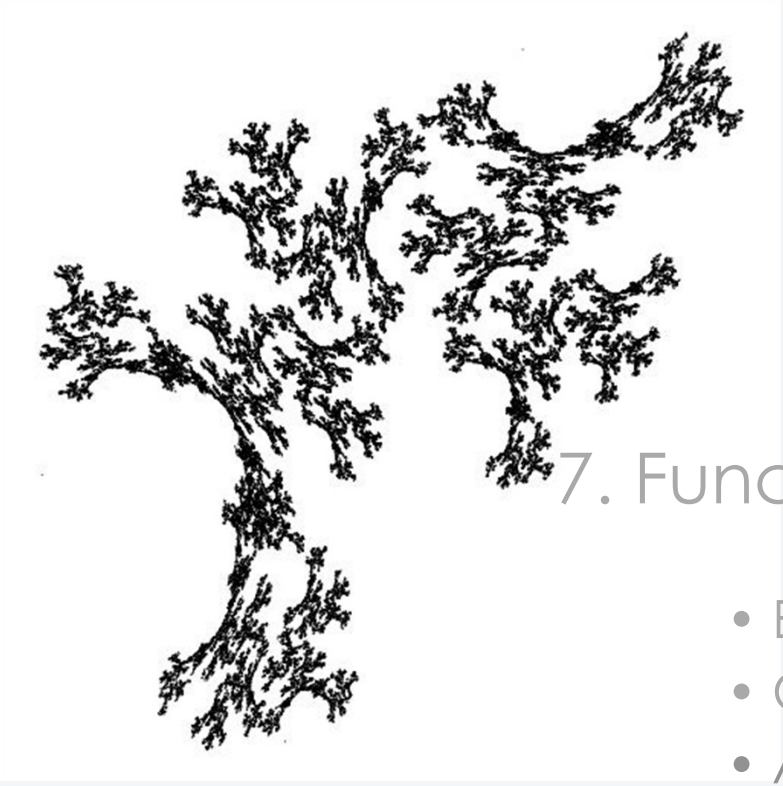
Example. Draw a plot of $\phi(x, 0, 1)$ in (–4, 4)

```
% java GaussianPlot 200
```




```
public class GaussianPlot
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      StdDraw.setXscale(-4.0, +4.0);
      StdDraw.setYscale(0, .5);
      StdDraw.setPenRadius(0.01);
      double[] x = new double[N+1];
      double[] y = new double[N+1];
      for (int i = 0; i <= N; i++)
      {
         x[i] = -4.0 + 8.0 * i / N;
         y[i] = Gaussian.pdf(x[i]);
      }
      for (int i = 0; i < N; i++)
         StdDraw.line(x[i], y[i], x[i+1], y[i+1]);
   }
}
```

Libraries of functions provide an easy way for *any* user (you) to extend the Java system.

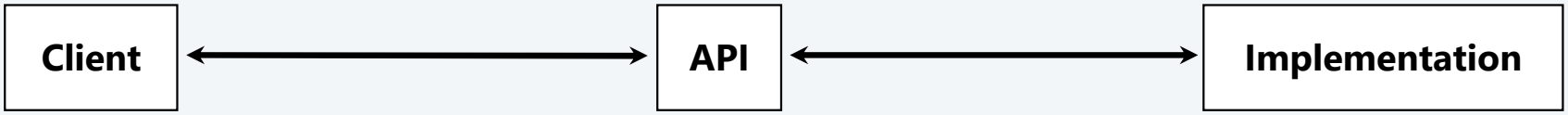


INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

7. Functions and Modules

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

Fundamental abstractions for modular programming



Client

Module that calls a library's methods.

Applications programming interface (API)

Defines signatures, describes methods.

Implementation

Module containing library's Java code.

```
public class GaussianPlot
{
  ...
  y[i] = Gaussian.pdf(x[i]);
  ...
}
```

public class Gaussian	
double pdf(double x)	*Gaussian probability density function*
double cdf(double x)	*Gaussian cumulative distribution function*

```
public class Gaussian
{
  public static double pdf(double x)
  {
    double val = Math.exp(-x*x / 2);
    val /=  Math.sqrt(2 * Math.PI);
    return val
  }
  ...
}
```

Example: **StdRandom** library

Developed for textbook, but broadly useful

- Implement methods for generating random numbers of various types.
- Available for download at booksite (and included in introcs software).

public class StdRandom	
int uniform(int N)	*integer between* 0 *and* N-1
double uniform(double lo, double hi)	*real between* lo *and* hi
booean bernoulli(double p)	true *with probability* p
double gaussian()	*normal with mean* 0, *stddev* 1
double gaussian(double m, double s)	*normal with mean* m, *stddev* s
int discrete(double[] a)	i *with probability* a[i]
void shuffle(double[] a)	*randomly shuffle the array* a[]

```
int getRandomNumber()
{
   return 4;  // chosen by fair dice roll.
              // guaranteed to be random.
}
```

First step in developing a library: Articulate the API!

StdRandom details

Implementation

```
public class StdRandom
{
   public static double uniform(double a, double b)
   {  return a + Math.random() * (b-a);  }

   public static int uniform(int N)
   {  return (int) (Math.random() * N);  }

   public static boolean bernoulli(double p)
   {  return Math.random() < p;  }

   public static double gaussian()
      /* see Exercise 1.2.27 */

   public static double gaussian(double m, double s)
   {  return mean + (stddev * gaussian());  }
   ...
}
```

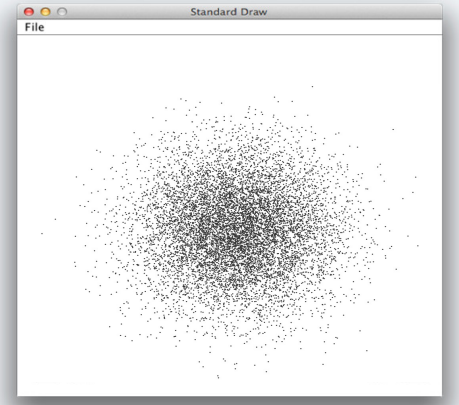
You *could* implement many of these methods,

but now you don't have to!

Typical client

```
public class RandomPoints
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 0; i < N; i++)
      {
         double x = StdRandom.gaussian(0.5, 0.2);
         double y = StdRandom.gaussian(0.5, 0.2);
         StdDraw.point(x, y);
      }
   }
}
```

```
% java RandomPoints 10000
```



Best practices

Small modules

- Separate and classify small tasks.
- Implement a layer of abstraction.

Independent development

- Code client *before* coding implementation.
- Anticipate needs of future clients.

```
public class StdRandom
{
   ...
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 0; i < N; i++) {
         StdOut.printf(" %2d " , uniform(100));
         StdOut.printf("%8.5f ", uniform(10.0, 99.0));
         StdOut.printf("%5b "  , bernoulli(.5));
         StdOut.printf("%7.5f ", gaussian(9.0, .2));
         StdOut.println();
      }
   }
}
```

Test clients

- Include main() test client in each module. ⟵ run all code at least once!
- Do more extensive testing in a separate module.

```
% java StdRandom 5
 61 21.76541  true 9.30910
 57 43.64327 false 9.42369
 31 30.86201  true 9.06366
 92 39.59314  true 9.00896
 36 28.27256 false 8.66800
```

Example: **StdStats** library

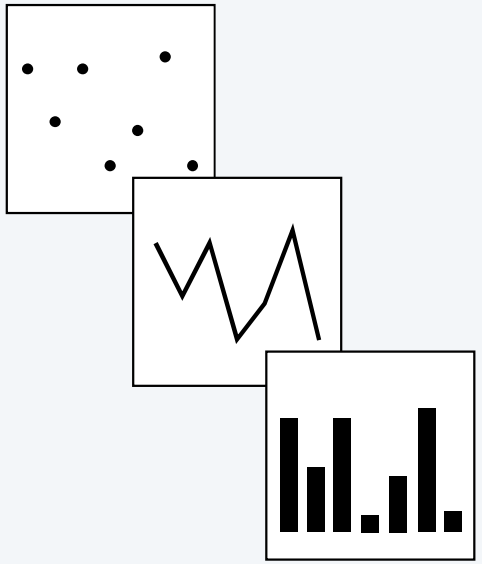
Developed for the textbook, but broadly useful

- Implement methods for computing statistics on arrays of real numbers.
- Available for download at booksite (and included in introcs software).

and plotting on StdDraw

public class StdStats	
double max(double[] a)	*largest value*
double min(double[] a)	*smallest value*
double mean(double[] a)	*average*
double var(double[] a)	*sample variance*
double stddev(double[] a)	*sample standard deviation*
double median(double[] a)	*plot points at* (i, a[i])
void plotPoints(double[] a)	*plot points at* (i, a[i])
void plotLines(double[] a)	*plot lines connecting points at* (i, a[i])
void plotBars(double[] a)	*plot bars to points at* (i, a[i])

API



Easy to implement, but easier to use!

one reason to develop a library: clarify client code

Example of modular programming: StdStats, StdRandom, and Gaussian client

Experiment

- Flip *N* coins.
- How many heads?
- Prediction: Expect *N*/2.

Prediction (more detailed)

- Run experiment *trials* times.
- How many heads?

```
public static int binomial(int N)
{
   int heads = 0;
   for (int j = 0; j < N; j++)
      if (StdRandom.bernoulli(0.5))
         heads++;
   return heads;
}
```

Goal. Write a program to validate predictions.

Example of modular programming: Bernoulli trials

```
public class Bernoulli
{
   public static int binomial(int N)
   // See previous slide.

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int trials = Integer.parseInt(args[1]);

      int[] freq = new int[N+1];
      for (int t = 0; t < trials; t++)
         freq[binomial(N)]++;

      double[] normalized = new double[N+1];
      for (int i = 0; i <= N; i++)
         normalized[i] = (double) freq[i] / trials;
      StdStats.plotBars(normalized);

      double mean = N / 2.0;
      double stddev = Math.sqrt(N) / 2.0;
      double[] phi  = new double[N+1];
      for (int i = 0; i <= N; i++)
         phi[i] = Gaussian.pdf(i, mean, stddev);
      StdStats.plotLines(phi);
   }
}
```

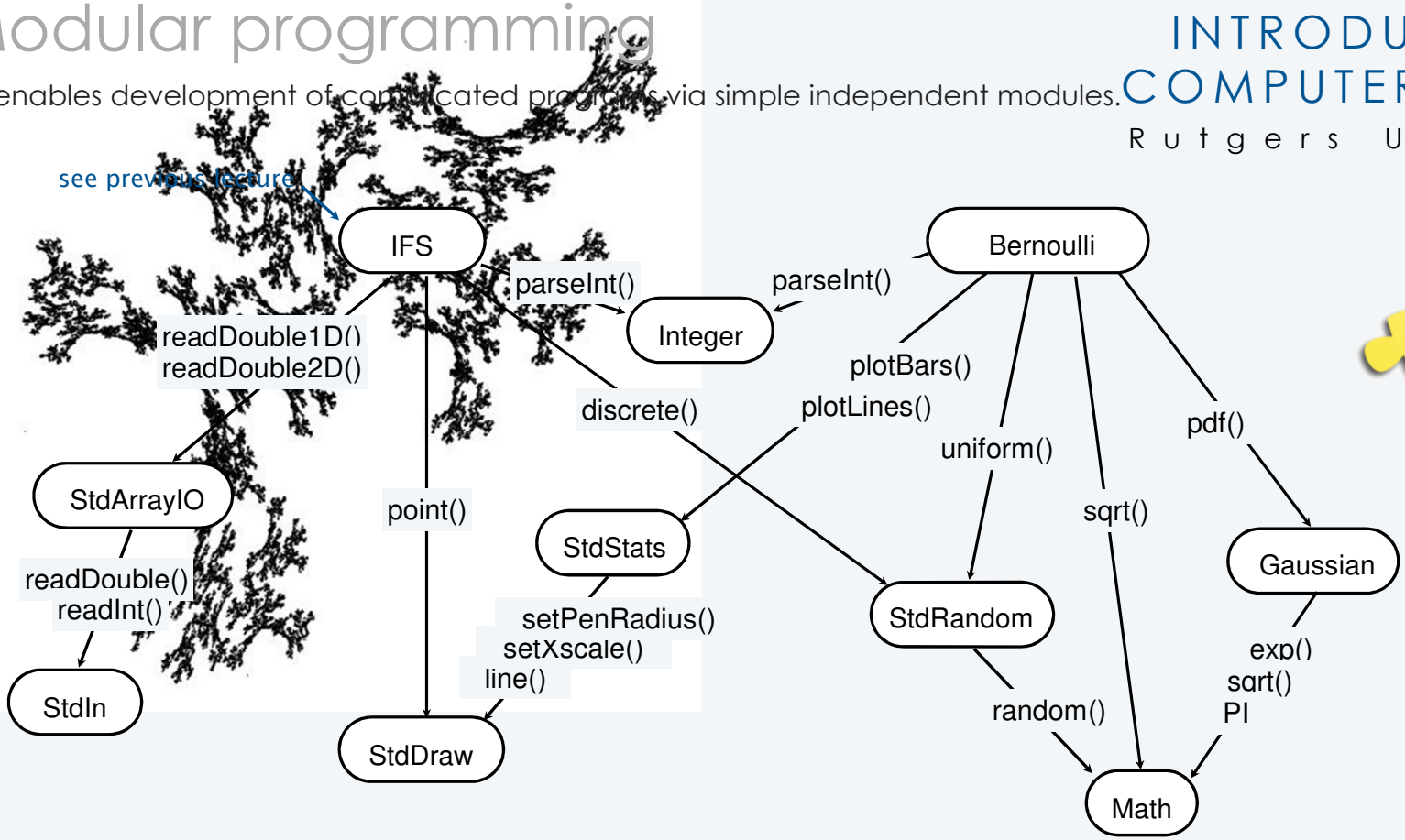
Bernoulli simulation

- Get command-line arguments (*trials* experiments of *N* flips).
- Run experiments. Keep track of frequency of occurrence of each return value.
- Normalize to between 0 and 1. Plot histogram.
- Plot theoretical curve.

```
% java Bernoulli 20 10000
```

Modular programming

- enables development of complicated programs via simple independent modules.



Advantages. Code is easier to understand, debug, maintain, improve, and reuse.

Why modular programming?

Modular programming enables

- Independent development of small programs.
- Every programmer to develop and share layers of abstraction.
- Self-documenting code.

Fundamental characteristics

- Separation of client from implementation benefits all *future* clients.
- Contract between implementation and clients (API) benefits all *past* clients.

Challenges

- How to break task into independent modules?
- How to specify API?

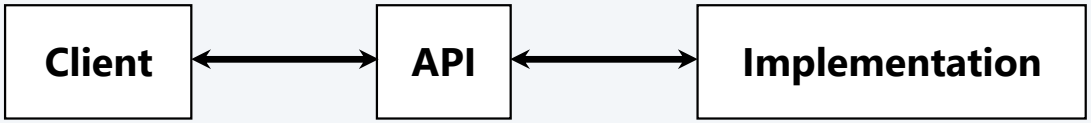


Image sources

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

Image sources

http://xkcd.com/221/

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

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

Image sources

http://commons.wikimedia.org/wiki/File:1405_Sound_Waves_and_the_Ear.jpg

http://en.wikipedia.org/wiki/The_Entertainer_(rag)#mediaviewer/File:EntertainerJoplinCover.JPEG

http://cantorion.org/music/497/The-Entertainer-Original-version

Image sources

http://www.intechopen.com/books/rheology-new-concepts-applications-and-methods/a-practical-review-of-microrheological-techniques

http://tweezpal.natan.si

http://www.albartlett.org/articles/art2000jan.html

http://laser.physics.sunysb.edu/~vwang/beam-propagation/

http://www.nature.com/neuro/journal/v14/n12/full/nn.2958.html

http://www-cdf.fnal.gov/physics/new/top/2012/WHel9ifb/

http://en.wikipedia.org/wiki/Deutsche_Mark#mediaviewer/File:DEU-10m-anv.jpg

INTRODUCTION TO
COMPUTER SCIENCE

Rutgers University

7. Functions and Modules

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

INTRODUCTION TO
COMPUTER SCIENCE

Rutgers University

7. Functions and Modules

http://introcs.cs.rutgers.edu