5. Functions and Libraries

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

Context: basic building blocks for programming

Any program you might want to write
- objects
- functions and libraries
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- Math
- text I/O
- primitive data types
- assignment statements

This lecture: Reuse code to build big programs from small pieces

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.
Def. A module is a .java file.

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).

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().

Java functions are more general than mathematical functions.

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.

```
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;
}
```

body of sqrt()
return statement
method's signature
argument declarations
method name
return type

Anatomy of a Java library

A library is a set of functions.

```
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));
    }
}
```

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().

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

```java
public class Newton {
    public static double sqrt(double c, double eps) {
        double t = c;
        while (Math.abs(t - c/t) > eps * t) {
            t = (t + c/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]);
        }
        double x = sqrt(a[0], 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).

**Note.** OS calls `main()` on Java command.

### Function Call Flow of Control Trace

```java
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 = (t + c/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]);
        }
        double x = sqrt(a[0], 1e-3);
        StdOut.println(x);
    }
}
```

<table>
<thead>
<tr>
<th>C</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a[i]</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1.414</td>
</tr>
<tr>
<td>2</td>
<td>1.732</td>
</tr>
<tr>
<td>3</td>
<td>1.732</td>
</tr>
</tbody>
</table>

**Warning:** Other methods used in other systems.

### Pop Quiz 1a on Functions

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

```java
public class PQFunctions1a {
    public static int cube(int i) {
        int j = 1 * 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(1));
    }
}
```

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

```
% javac PQfunctions1a.java
% java PQfunctions1a 6
3
27
64
125
216
```
Pop quiz 1b on functions

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

```java
public class PQFunctions1b {
    public static int cube(int i) {
        int t = i * i * i;
        return t;
    }
    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:1: i is already defined in cube(int)
  int t = i * i * i;
  ^
1 error
```

Pop quiz 1c on functions

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

```java
public class PQFunctions1c {
    public static int cube(int i) {
        int t = 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?

```java
public class PQfunctionsId {
    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.

Pop quiz 1e on functions

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

```java
public class PQfunctionsIe {
    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.
5. Functions and Libraries

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

Digital audio

To represent a wave, \textit{sample} at regular intervals and save the values in an array.

<table>
<thead>
<tr>
<th>samples/sec</th>
<th>samples</th>
<th>sampled waveform</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,512</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>11,025</td>
<td>275</td>
<td></td>
</tr>
<tr>
<td>22,050</td>
<td>551</td>
<td></td>
</tr>
<tr>
<td>CD standard</td>
<td>44,100</td>
<td>1102</td>
</tr>
</tbody>
</table>

Bottom line. You can \textit{write programs} to manipulate sound (arrays of double values).

Crash course in sound

\textbf{Sound} is the perception of the vibration of molecules.

A \textbf{musical tone} is a steady periodic sound.

A \textbf{pure tone} is a sinusoidal waveform.

Western musical scale
• Concert A is 440 Hz.
• 12 notes, logarithmic scale.

\begin{tabular}{|c|c|c|}
\hline
\textbf{pitch} & \textbf{i} & \textbf{frequency (440^{2i})} & \textbf{sinusoidal waveform} \\
\hline
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 & 622.55 & \\
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 & \\
\hline
\end{tabular}
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.

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 duration = Double.parseDouble(args[1]);
        double[] a = tone(hz, duration);
        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 * 1.1);
        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?

```java
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
% java PQfunctions2.java
```

---

Play that chord

Produce chords by *averaging* waveforms.

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

---

Play that chord implementation

```java
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 h2 = 440.0 * Math.pow(2, pitch1 / 12.0);
            double h2 = 440.0 * Math.pow(2, pitch2 / 12.0);
            double[] a = PlayThatNote.tone(h2, d);
            double[] b = PlayThatNote.tone(h2, 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);
        }
    }
}
```

---

Play that tune (deluxe version)

Add harmonics to PlayThatTune to produce a more realistic sound.

```java
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[] b = tone(2.0 * hz, duration);
    double[] hi = tone(0.5 * hz, duration);
    double[] harmonic = sum(h1, h2);
    return avg(a, harmonic);
}
```

Function to add harmonics to a tone

```java
double harmonic = sum(h1, h2);
```

```java
% java PlayThatChord 0 1 5.0
% java PlayThatChord 0 12 5.0
% java PlayThatTune 1.5 - elise.txt
```

Program 2.14 in text (with tempo added)
Digital audio (summary)

Bottom line. You can write programs to manipulate sound.
This lecture: Case study of the utility of functions.
Upcoming assignment: Fun with musical tones.

5. Functions and Libraries

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

Gaussian distribution

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

Polystyrene particles in glycerol

Calibration of optical tweezers

Laser beam propagation

Polarized W bosons from top-quark decay

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.

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

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 \( \Phi(820, 1019, 209) = 0.17050966869132111... \)

Defining a library of functions

Q. Is the Gaussian pdf \( \Phi \) implemented in Java's Math library?
A. No.

Q. Why not?
A. Maybe because it is so easy for you to do it yourself.

```java
public class Gaussian
{
    public static double pdf(double x) { return Math.exp(-Math.pow(x, 2) / 2) / Math.sqrt(2 * Math.PI); }
    public static double pdf(double x, double mu, double sigma) { return pdf(x - mu) / (sigma); }
    // Stay tuned for more functions.
}
```

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

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

A. Use Taylor series. \( Φ(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \, dx = \frac{1}{2} + \sum_{n=1}^{\infty} \frac{z^n}{n!} \left( z^2 + \frac{z^4}{3} + \frac{z^6}{5} + \frac{z^8}{7} + \ldots \right) \)

```java
public static double cdf(double z) {
    double sum = 0.0; 
    term = z; 
    for (int i = 1; sum + term != sum; i *= 2) 
        sum += term; 
    term = term * z * z / i; 
    return 0.5 + sum * pdf(z); 
}
```

```java
public static double cdf(double z, double mu, double sigma) { 
    return cdf(z-mu) / sigma; 
}
```

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
X: Gaussian 820 1019 209
0.170509685910111
```

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.

Using a library

To use these methods in another program

• 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
x: GaussianPlot 200
```

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

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

Fundamental abstractions for modular programming

Client
- Module that calls a library's methods.

API
- Defines signatures, describes methods.

Implementation
- Module containing library's Java code.

Example: StdRandom library

Developed for this course, but broadly useful
- Implement methods for generating random numbers of various types.
- Available for download at booksite (and included in introcs software).

Typical client

You could implement many of these methods, but now you don't have to!
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.

Test clients
• Include main() test client in each module.
• Do more extensive testing in a separate module.

Example: StdStats library

Developed for this course, but broadly useful
• Implement methods for computing statistics on arrays of real numbers.
• Available for download at booksite (and included in intros software).

public class StdStats
{
    double max(double[] a) { return a[0]; }  \largest value
    double min(double[] a) { return a[0]; }  \smallest value
    double mean(double[] a) { return a[0]; } \average
    double var(double[] a) { return a[0]; } \sample variance
    double stddev(double[] a) { return a[0]; } \sample standard deviation
    double median(double[] a) { return a[0]; } \plot points at (i, a[i])
    void plotPoints(double[] a) { return a[0]; } \plot lines connecting points at (i, a[i])
    void plotBars(double[] a) { return a[0]; } \plot bars to points at (i, a[i])
}

API

Example of modular programming: Bernoulli trials

public class Bernoulli
{
    public static int binomial(int N) { return N; } \binomial distribution
    public static int binomial(int N) { return N; } \binomial distribution
    double mean = N / 2.0; \double mean = N / 2.0;
    double stddev = Math.sqrt(N) / 2.0; \double stddev = Math.sqrt(N) / 2.0;
    double[] phi = new double[N+2]; \phi[i] = Gaussian.pdf(i, mean, stddev);
    for (int i = 0; i < N; i++) \phi[i] = Gaussian.pdf(i, mean, stddev);
    phi[N] = Gaussian.pdf(N, mean, stddev);
    StdStats.plotPoints(phi); \plot points at (i, a[i])
    StdStats.plotLines(phi); \plot lines connecting points at (i, a[i])
    StdStats.plotBars(phi); \plot bars to points at (i, a[i])
}

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.
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?