2.1, 2.2 Functions and Libraries

input number

output number

\[ f(x, y, z) = x \times 2 + 5 \]
A Foundation for Programming

- objects
- functions and modules
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- Math
- text I/O
- primitive data types
- assignment statements

Any program you might want to write can build bigger programs and reuse code.
Functions (Static Methods)

Java function.
- 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.
- Our I/O libraries: `StdIn.readInt()`, `StdDraw.line()`, `StdAudio.play()`.
- User-defined functions: `main()`.
Anatomy of a Java Function

Java functions. Easy to write your own.

\[ f(x) = \sqrt{x} \]

input: 2.0 → \( f(x) = \sqrt{x} \) → output: 1.414213...

```
public static double sqrt ( double c )
{
    if (c < 0) return Double.NaN;
    double err = 1e-15;
    double t = c;
    while (Math.abs(t - c/t) > err * t)
        t = (c/t + t) / 2.0;
    return t;
}
```
public class Gambler {
    public static void main(String[] args) {
        int stake   = Integer.parseInt(args[0]);
        int goal    = Integer.parseInt(args[1]);
        int trials  = Integer.parseInt(args[2]);
        . . .
    }
}
Key point. Functions provide a new way to control the flow of execution.

```java
public class Newton {
    public static double sqrt(double c) {
        double epsilon = 1e-15;
        if (c < 0) return Double.NaN;
        double t = c;
        while (Math.abs(t - c/t) > epsilon * 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]);
            StdOut.println(x);
        }
    }
}
```
Key point. Functions provide a new way to control the flow of execution.

Summary of what happens when a function is called:

- Control transfers to the function code.
- Argument variables are assigned the values given in the call.
- Function code is executed.
- Return value is assigned in place of the function name in the calling code.
- Control transfers back to the calling code.

Note. This technique (standard in Java) is known as “pass by value”.

other languages may use different methods
**Scope**

**Scope (of a name).** The code that can refer to that name.

**Def.** A variable’s scope is code following the declaration in its block.

```java
public class Newton {
    public static double sqrt(double c) {
        double epsilon = 1e-15;
        if (c < 0) return Double.NaN;
        double t = c;
        while (Math.abs(t - c/t) > epsilon * 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++) {
            System.out.println(sqrt(a[i]));
        }
    }
}
```

**Best practice:** declare variables so as to **limit** their scope.
public class Newton
{
    public static double sqrt(double c)
    {
        double epsilon = 1e-15;
        if (c < 0) return Double.NaN;
        double t = c;
        while (Math.abs(t - c/t) > epsilon * 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++)
            System.out.println(sqrt(a[i]));
    }
}
public class Newton {
    public static double sqrt(double c) {
        double epsilon = 1e-15;
        if (c < 0) return Double.NaN;
        double t = c;
        while (Math.abs(t - c / t) > epsilon * 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++)
            System.out.println(sqrt(a[i]));
    }
}

Find this great tool on the Precepts page.

Copy code into it from slide, book site, or your own code.
What happens when you compile and run the following code?

```java
public class Cubes1 {
    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));
    }
}
```
What happens when you compile and run the following code?

```java
public class Cubes2 {
    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));
    }
}
```
What happens when you compile and run the following code?

```java
public class Cubes3 {
    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));
    }
}
```
What happens when you compile and run the following code?

```java
public class Cubes4 {
    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));
    }
}
```
What happens when you compile and run the following code?

```java
public class Cubes5 {
    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));
    }
}
```
Example: Gaussian Distribution
Gaussian Distribution

Standard Gaussian distribution.

- "Bell curve."
- Basis of most statistical analysis in social and physical sciences.

**Ex.** 2000 SAT scores follow a Gaussian distribution with mean $\mu = 1019$, stddev $\sigma = 209$.

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

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

\[= \phi\left(\frac{x-\mu}{\sigma}\right) / \sigma\]
Java Function for $\phi(x)$

Mathematical functions. Use built-in functions when possible; build your own when not available.

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

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

Overloading. Functions with different signatures are different.

Multiple arguments. Functions can take any number of arguments.

Calling other functions. Functions can call other functions.
Gaussian Cumulative Distribution Function

**Goal.** Compute Gaussian cdf $\Phi(z)$.

**Challenge.** No "closed form" expression and not in Java library.

\[
\Phi(z) = \int_{-\infty}^{z} \phi(x) dx
\]

Taylor series

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

Bottom line. 1,000 years of mathematical formulas at your fingertips.
public class Gaussian
{
    public static double phi(double x)
        // as before
    
    public static double Phi(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 * phi(z);
    }

    public static double Phi(double z, double mu, double sigma)
    {
        return Phi((z - mu) / sigma);
    }
}

accurate with absolute error
less than 8 * 10^{-16}

$$\Phi(z, \mu, \sigma) = \int_{-\infty}^{z} \phi(z, \mu, \sigma) = \Phi((z - \mu) / \sigma)$$
Q. NCAA requires at least 820 for Division I athletes. What fraction of test takers in 2000 did not qualify?

A. $\Phi(820, \mu, \sigma) \approx 0.17051$. [approximately 17%]

double fraction = Gaussian.Phil(820, 1019, 209);
Q. Why relevant in mathematics?
A. Central limit theorem: under very general conditions, average of a set of variables tends to the Gaussian distribution.

Q. Why relevant in the sciences?
A. Models a wide range of natural phenomena and random processes.
   - Weights of humans, heights of trees in a forest.
   - SAT scores, investment returns.

Caveat.

Everybody believes in the exponential law of errors: the experimenters, because they think it can be proved by mathematics; and the mathematicians, because they believe it has been established by observation. - M. Lippman in a letter to H. Poincaré
**Library.** A module (class) whose methods are primarily intended for use by many other programs.

**Client.** Program that calls library method(s).

**API.** Contract between client and implementation.

**Implementation.** Program that implements the methods of an API (i.e., contains the code).
Libraries

Why use libraries?

• **Makes code easier to understand.**
• **Makes code easier to debug.**
• **Makes code easier to maintain and improve.**
• ** Makes code easier to reuse.**
Digital Audio
Crash Course in Sound

**Sound.** Perception of the **vibration** of molecules in our eardrums.

**Concert A.** Sine wave, scaled to oscillate at 440Hz.

**Other notes.** 12 notes on chromatic scale, divided logarithmically.

<table>
<thead>
<tr>
<th>note</th>
<th>i</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>440.00</td>
</tr>
<tr>
<td>A♯ or B♭</td>
<td>1</td>
<td>466.16</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>493.88</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>523.25</td>
</tr>
<tr>
<td>C♯ or D♭</td>
<td>4</td>
<td>554.37</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>587.33</td>
</tr>
<tr>
<td>D♯ or E♭</td>
<td>6</td>
<td>622.25</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>659.26</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>698.46</td>
</tr>
<tr>
<td>F♯ or G♭</td>
<td>9</td>
<td>739.99</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>783.99</td>
</tr>
<tr>
<td>G♯ or A♭</td>
<td>11</td>
<td>830.61</td>
</tr>
<tr>
<td>A</td>
<td>12</td>
<td>880.00</td>
</tr>
</tbody>
</table>

Notes, numbers, and waves

\[ 440 \times 2^{\frac{i}{12}} \]
Digital Audio

**Sampling.** Represent curve by sampling it at regular intervals.

\[
y(i) = \sin \left( \frac{2\pi \cdot i \cdot 440}{44,100} \right)
\]

- 5,512 samples/second, 137 samples
- 11,025 samples/second, 275 samples
- 22,050 samples/second, 551 samples
- 44,100 samples/second, 1,102 samples

audio CD
Musical tone. Create a music tone of a given frequency and duration.

```java
public class Tone {
    public static void main(String[] args) {
        int sps = 44100;
        double hz       = Double.parseDouble(args[0]);
        double duration = Double.parseDouble(args[1]);
        int N = (int) (sps * duration);
        double[] a = new double[N+1];
        for (int i = 0; i <= N; i++)
            a[i] = Math.sin(2 * Math.PI * i * hz / sps);
        StdAudio.play(a);
    }
}
```

\[ y(i) = \sin \left( \frac{2\pi \cdot i \cdot hz}{44,100} \right) \]

% java Tone 440 1.5
[ concert A for 1.5 seconds]
Play That Tune

**Goal.** Play pitches and durations from standard input on standard audio.

```java
public class PlayThatTune
{
    public static void main(String[] args)
    {
        int sps = 44100;
        while (!StdIn.isEmpty())
        {
            int pitch = StdIn.readInt();
            double duration = StdIn.readDouble();
            double hz = 440 * Math.pow(2, pitch / 12.0);
            int N = (int) (sps * duration);
            double[] a = new double[N+1];
            for (int i = 0; i <= N; i++)
                a[i] = Math.sin(2 * Math.PI * i * hz / sps);
            StdAudio.play(a);
        }
    }
}
```

% java PlayThatTune < elise.txt
Musical Tone Function

Musical tone. Create a music tone of a given frequency and duration.

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

Remark. Can use arrays as function return value and/or argument.
Standard audio. Library for playing digital audio.

```java
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 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
    }
}
```

**Concert A.** Play concert A for 1.5 seconds using `StdAudio`.

```java
double[] a = tone(440, 1.5);
StdAudio.play(a);
```

**Remark.** Java arrays passed “by reference” (no copy made).
**Concert A with harmonics.** Obtain richer sound by adding tones one octave above and below concert A.

- **880 Hz**
- **220 Hz**
- **440 Hz**

---

\[\text{lo} = \text{tone}(220, .0041); \]
\[\text{lo}[44] = .982\]

\[\text{hi} = \text{tone}(880, .0041); \]
\[\text{hi}[44] = -.693\]

\[\text{h} = \text{sum}(\text{hi}, \text{lo}, .5, .5); \]
\[\text{h}[44] = .5\times\text{lo}[44] + .5\times\text{hi}[44]; \]
\[= .5\times.982 - .5\times.693 = .144\]

\[\text{A} = \text{tone}(440, .0041); \]
\[\text{A}[44] = .374\]

\[\text{sum}(\text{A}, \text{h}, .5, .5); \]
\[\text{A}[44] + \text{h}[44] = .5\times.144 + .5\times.374 \]
\[= .259\]
public class PlayThatTuneDeluxe                   // improved version with Harmonics
{
    // Return weighted sum of two arrays.
    public static double[] sum(double[] a, double[] b, double awt, double bwt) {
        double[] c = new double[a.length];
        for (int i = 0; i < a.length; i++)
            c[i] = a[i]*awt + b[i]*bwt;
        return c;
    }

    // Return a note of given pitch and duration.
    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[] h = sum(hi, lo, .5, .5);
        return sum(a, h, .5, .5);
    }

    public static double[] tone(double hz, double t)
        // see previous slide
    
    public static void main(String[] args)
        // see next slide
    }
Harmonics

Play that tune (deluxe version). Read in pitches and durations from standard input, and play using standard audio.

```java
public static void main(String[] args) {
    while (!StdIn.isEmpty()) {
        int pitch = StdIn.readInt();
        double duration = StdIn.readDouble();
        double[] a = note(pitch, duration);
        StdAudio.play(a);
    }
}
```

% more elise.txt  % java PlayThatTune < elise.txt
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25
public class PlayThatTune
{
    public static double[] sum(double[] a, double[] b, double avt, double bwt)
    {
        double[] c = new double[a.length];
        for (int i = 0; i < a.length; i++)
            c[i] = a[i]*avt + b[i]*bwt;
        return c;
    }

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

    public static double[] note(int pitch, double t)
    {
        double hz = 440.0 * Math.pow(2, pitch / 12.0);
        double[] a = tone(hz, t);
        double[] hi = tone(2*hz, t);
        double[] lo = tone(hz/2, t);
        double[] h = sum(hi, lo, .5, .5);
        return sum(a, h, .5, .5);
    }

    public static void main(String[] args)
    {
        while (!StdIn.isEmpty())
        {
            int pitch = StdIn.readInt();
            double duration = StdIn.readDouble();
            double[] a = note(pitch, duration);
            StdAudio.play(a);
        }
    }
}