2.1 Functions

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
build bigger programs and reuse code

Functions (Static Methods)

Java function.
- Takes zero or more input arguments.
- Returns one output value.
- 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 \[ 2.0 \]  
output \[ 1.414213 \ldots \]

Scope

Scope (of a name). The code that can refer to that name. 
Ex. A variable's scope is code following the declaration in the block.

```
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]);
        double x = sqrt(a[1]);
        StdOut.println(x);
    }
}
```

Best practice: declare variables to limit their scope.

Flow of Control

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

```
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) {
        int N = args.length;
        double[] a = new double[N];
        for (int i = 0; i < N; i++)
            a[i] = Double.parseDouble(args[i]);
        double x = sqrt(a[1]);
        StdOut.println(x);
    }
}
```

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

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 calling code.
- Control transfers back to the calling code.

Note. This is known as "pass by value."
Function Examples

<table>
<thead>
<tr>
<th>absolute value of an int value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static int abs(int x)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>if (x &lt; 0) return -x;</td>
</tr>
<tr>
<td>else return x;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>absolute value of a double value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static double abs(double x)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>if (x &lt; 0.0) return -x;</td>
</tr>
<tr>
<td>else return x;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>primality test</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static boolean isPrime(int N)</td>
</tr>
<tr>
<td>{   if (N &lt; 2) return false;</td>
</tr>
<tr>
<td>for (int i = 2; i &lt;= N; i++)</td>
</tr>
<tr>
<td>if (N % i == 0) return false;</td>
</tr>
<tr>
<td>return true;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hypotenuse of a right triangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static double hypotenuse(double a, double b)</td>
</tr>
<tr>
<td>{    return Math.sqrt(a<em>a + b</em>b);  }</td>
</tr>
</tbody>
</table>

Function Challenge 1a

**Q.** 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));
    }
}
```

% javac Cubes1.java
% java Cubes1 6
1 1
2 8
3 27
4 64
5 125
6 216

Function Challenge 1b

**Q.** 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));
    }
}
```

Function Challenge 1c

**Q.** 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));
    }
}
```
Q. 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));
    }
}
```

Q. 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));
    }
}
```

---

**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 $\sigma = 209$.

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

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

```java
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 $8 \times 10^{-16}$

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

SAT Scores

**Q.** NCAA requires at least 820 for Division I athletes. What fraction of test takers in 2000 do not qualify?

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

```java
double fraction = Gaussian.Phi(820, 1019, 209);
```
Gaussian Distribution

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

Building Functions

Functions enable you to build a new layer of abstraction.
  - Takes you beyond pre-packaged libraries.
  - You build the tools you need: Gaussian.phi(), ...

Process.
  - Step 1: identify a useful feature.
  - Step 2: implement it.
  - Step 3: use it.
  - Step 3': re-use it in any of your programs.

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#</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#</td>
<td>4</td>
<td>554.37</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>587.33</td>
</tr>
<tr>
<td>D#</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>698.45</td>
</tr>
<tr>
<td>F#</td>
<td>9</td>
<td>739.99</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>783.99</td>
</tr>
<tr>
<td>G#</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
Digital Audio

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

- 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

**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.

Digital Audio in Java

**Standard audio.** Library for playing digital audio.

```java
public class StdAudio {
    void play(String file) { /* play the given wave 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 wave file */ }
    void double[] read(String file) { /* read from a wave file */ }
}
```

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

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

Harmonics

**Concert A with harmonics.** Obtain richer sound by adding tones one octave above and below concert A.

- 880 Hz
- 220 Hz
- 440 Hz

- \( y(t) = \sin \left( \frac{2\pi \cdot i \cdot 440}{44100} \right) \)
`public class PlayThatTune {

    // 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
    }
}`

**Play that tune.** 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
7 .125
7 .125
7 .125
2 .125
5 .125
3 .125
0 .25

---

### 2.2 Libraries and Clients

Introduction to Programming in Java: An Interdisciplinary Approach
· Robert Sedgewick and Kevin Wayne
· Copyright © 2008 · February 22, 2009 3:09 AM

[Cover Image: Introduction to Programming in Java: An Interdisciplinary Approach]
**Libraries**

**Library.** A module whose methods are primarily intended for use by many other programs.

**Client.** Program that calls a library.

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

**Implementation.** Program that implements the methods in an API.

---

**Random Numbers**

"The generation of random numbers is far too important to leave to chance. Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

---

**Standard Random**

**Standard random.** Our library to generate pseudo-random numbers.

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

    // between 0 and N-1
    public static int uniform(int N) {
        return (int)(Math.random() * N);
    }

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

    // gaussian with mean = 0, stddev = 1
    public static double gaussian() {
        // recall Assignment 0
        return Math.random(); // guaranteed to be random.
    }

    // gaussian with given mean and stddev
    public static double gaussian(double mean, double stddev) {
        return mean + (stddev * gaussian());
    }
}
```
**Unit Testing**

**Unit test. Include main() to test each library.**

```java
public class StdRandom {
    ...
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double[] t = {0.5, 0.3, 0.1, 0.1};
        for (int i = 0; i < N; i++) {
            StdOut.printf("%2d",
                StdRandom.universal(100));
            StdOut.printf("%8.5f",
                StdRandom.gaussian(10.0, 99.0));
            StdOut.printf("%5b",
                StdRandom.bernoulli(0.5));
            StdOut.printf("%7.5f",
                StdRandom.gaussian(9.0, 2));
            StdOut.printf("%2d",
                StdRandom.discrete(t));
            StdOut.println();
        }
    }
}
```

```
% java StdRandom 5
61 21.76541 true 9.30910 0
57 43.64327 false 9.42369 3
31 30.86201 true 9.06366 0
92 39.59314 true 9.00896 0
36 28.27256 false 8.66800 1
```

**Using a Library**

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

```
% javac RandomPoints.java
% java RandomPoints 10000
```

**Standard Statistics**

**Ex. Library to compute statistics on an array of real numbers.**

```java
public class StdStats {
    double max(double[] a) { return largest value; }
    double min(double[] a) { return smallest value; }
    double mean(double[] a) { return average; }
    double var(double[] a) { return sample variance; }
    double stddev(double[] a) { return sample standard deviation; }
    double median(double[] a) { return median; }
    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]); }

    double mu = \frac{a_0 + a_1 + \cdots + a_{n-1}}{n}, \quad \sigma^2 = \frac{(a_0 - \mu)^2 + (a_1 - \mu)^2 + \cdots + (a_{n-1} - \mu)^2}{n - 1}
    \text{mean} \quad \text{sample variance}
```

\[
\mu = \frac{a_0 + a_1 + \cdots + a_{n-1}}{n}, \quad \sigma^2 = \frac{(a_0 - \mu)^2 + (a_1 - \mu)^2 + \cdots + (a_{n-1} - \mu)^2}{n - 1}
\]

\text{mean} \quad \text{sample variance}
Ex. Library to compute statistics on an array of real numbers.

```java
public class StdStats {
    public static double max(double[] a) {
        double max = Double.NEGATIVE_INFINITY;
        for (int i = 0; i < a.length; i++)
            if (a[i] > max) max = a[i];
        return max;
    }

    public static double mean(double[] a) {
        double sum = 0.0;
        for (int i = 0; i < a.length; i++)
            sum = sum + a[i];
        return sum / a.length;
    }

    public static double stddev(double[] a) {
        // see text
    }
}
```

Modular Programming

Modular programming.
- Divide program into self-contained pieces.
- Test each piece individually.
- Combine pieces to make program.

Ex. Flip N coins. How many heads?
- Read arguments from user.
- Flip one fair coin.
- Flip N fair coins and count number of heads.
- Repeat simulation, counting number of times each outcome occurs.
- Plot histogram of empirical results.
- Compare with theoretical predictions.

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

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

        int[] freq = new int[N+1];
        for (int i = 0; i <= N; i++)
            freq[i] = (int) Math.pow(T, i) * Math.pow(1-T, N-i);

        double[] normalized = new double[N+1];
        for (int i = 0; i <= N; i++)
            normalized[i] = freq[i] / T;

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

        StdStats.plotBars(normalized);
        StdStats.plotLines(phi);
    }
}
```
**Modular programming.** Build relatively complicated program by combining several small, independent, modules.

**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.