2.1 Functions
2.1 Functions

\[ f(x, y, z) \]
A Foundation for Programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math

text I/O

primitive data types

assignment statements

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.
- 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 Function

Java functions. Easy to write your own.

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

\[ \text{input} \quad 2.0 \quad \rightarrow \quad \text{output} \quad 1.414213\ldots \]
Flow of Control

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

```java
public class Newton {
    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 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]);
        for (int i = 0; i < N; i++)
            {
                double x = sqrt(a[i]);
            }
        StdOut.println(x);
    }
}
```
Flow of Control

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."
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.

Best practice: declare variables to limit their scope.
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
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));
    }
}
```
Function Challenge 1d

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

```java
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));
    }
}
```
Function Challenge 1e

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
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;
    }
}
```

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

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

**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.

Bottom line. 1,000 years of mathematical formulas at your fingertips.
Java function for $\Phi(z)$

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

$\Phi(z, \mu, \sigma) = \int_{-\infty}^{z} \phi(z, \mu, \sigma) = \Phi((z-\mu) / \sigma)$
SAT Scores

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

A. \( \Phi(820, 1019, 209) \approx 0.17051. \) [approximately 17%]

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 random 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
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
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 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;
}

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

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 .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.

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

library developed for this course (also broadly useful)
Harmonics

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

880 Hz 220 Hz 440 Hz

\[ l_0 = \text{tone}(220, .0041); \]
\[ l_0[44] = .982 \]

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

\[ h = \text{sum}(h_i, l_0, .5, .5); \]
\[ h[44] = .5 \times l_0[44] + .5 \times h_i[44]; \]
\[ = .5 \times .982 - .5 \times .693 = .144 \]

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

\[ \text{sum}(A, h, .5, .5); \]
\[ A[44] + h[44] = .5 \times .144 + .5 \times .374 \]
\[ = .259 \]
public class PlayThatTuneDeluxe {

    // 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. 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
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25

% java PlayThatTune < elise.txt
public class PlayThatTune
{
    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;
    }

    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);
        }
    }
}
2.2 Libraries and Clients
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.
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.

public class StdRandom

    int uniform(int N)           // integer between 0 and N-1
    double uniform(double lo, double hi)  // real between lo and hi
    boolean bernoulli(double p)      // true with probability p
    double gaussian()               // normal, mean 0, standard deviation 1
    double gaussian(double m, double s)  // normal, mean m, standard deviation 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.
    }
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() /* see Exercise 1.2.27 */

// 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]);
        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();
        }
    }
}

% 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
```
Using a Library

```java
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);
        }
    }
}
```

% javac RandomPoints.java
% java RandomPoints 10000
Statistics
# Standard Statistics

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

```java
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) { /* 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]) */
```

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

*mean* \hspace{1cm} *sample variance*
Standard Statistics

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

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

   public 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 < T; i++)
         freq[binomial(N)]++;

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

Bernoulli Trials

The code above demonstrates how to simulate Bernoulli trials using the `StdRandom.bernoulli` function. It calculates the binomial distribution for a given number of trials `N` and performs `T` trials of `N` coin flips each, recording the number of heads for each possible outcome. The results are then plotted as a histogram to visualize the distribution. Additionally, the code computes the theoretical mean and standard deviation of the binomial distribution and plots a theoretical prediction line to compare with the simulation results.
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.