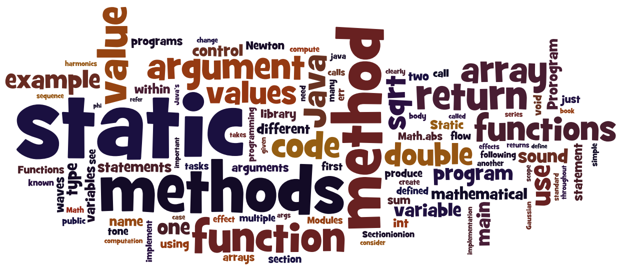
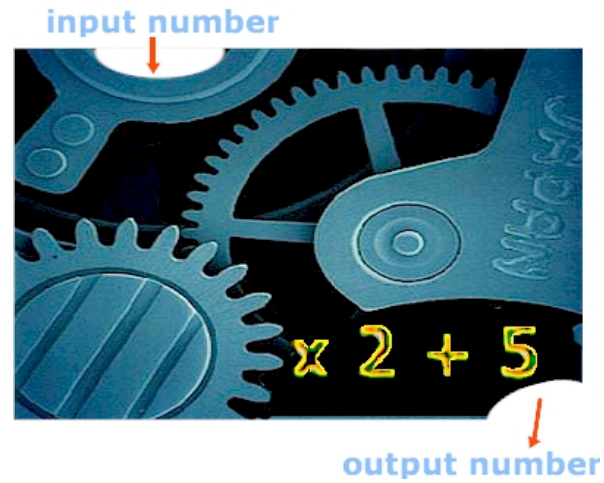
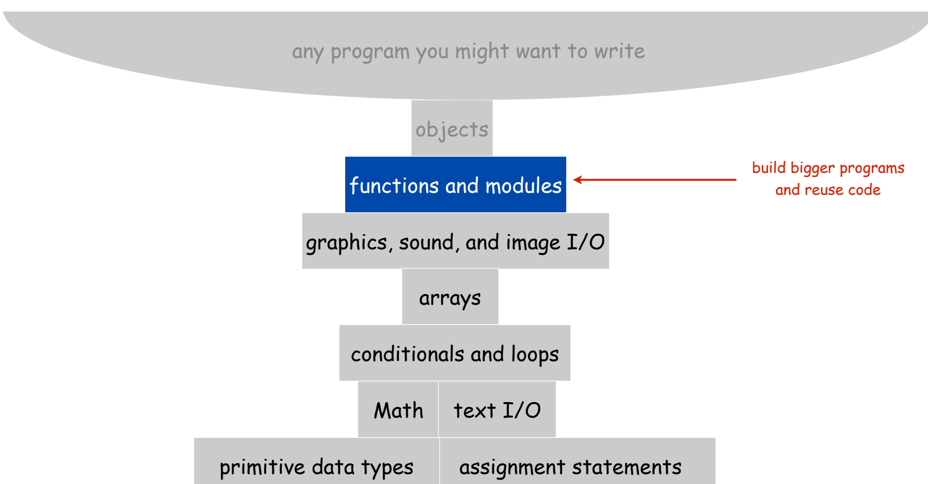


2.1 Functions



A Foundation for Programming



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).

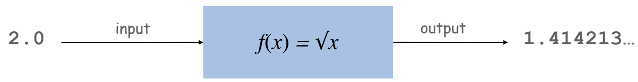
more general than mathematical functions

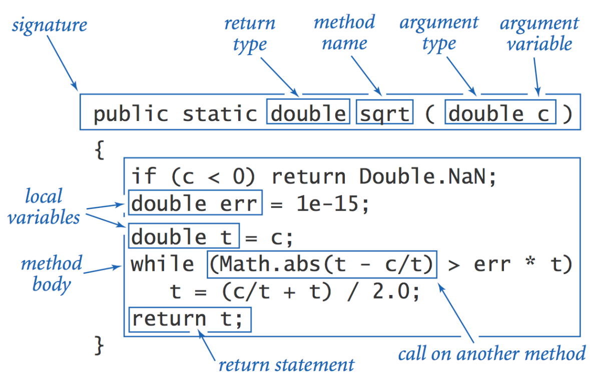
Applications.
- Scientists use mathematical functions to calculate formulas.
- Programmers use functions to build modular programs.
- You use functions for both.

Examples.
- 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.





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)
   {
      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);
      }
   }
}
```


Flow of Control

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 method (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.

```
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]));
   }
}
```

scope of c
scope of epsilon
scope of t
scope of a

two different variables with the same name i each with two lines of scope

Best practice: declare variables so as to limit their scope.

Function Call Trace

```
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]));
   }
}
```


TEQ on Functions 1.1

What happens when you compile and run the following code?

```
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));
   }
}
```


TEQ on Functions 1.2

What happens when you compile and run the following code?

```
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));
   }
}
```


TEQ on Functions 1.3

What happens when you compile and run the following code?

```
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));
   }
}
```

TEQ on Functions 1.4

What happens when you compile and run the following code?

```
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));
   }
}
```


TEQ on Functions 1.5

What happens when you compile and run the following code?

```
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));
   }
}
```

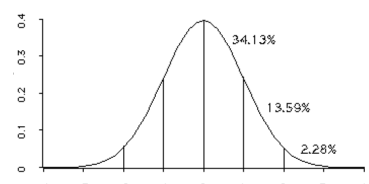

Example: Gaussian Distribution

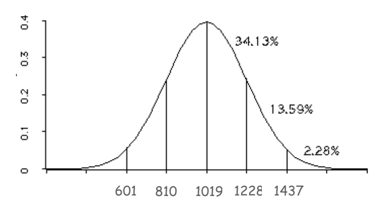
Gaussian Distribution

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 μ = 1019, stddev σ = 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.

```
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;
   }
}
```

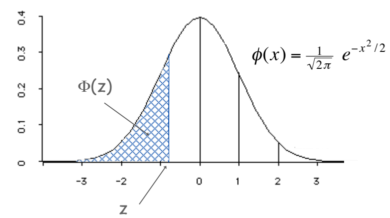
$$\phi(x) = \frac{1}{\sqrt{2\pi}}\, e^{-x^2/2}$$

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

Overloading. Functions with different signatures are different.
Multiple arguments. Functions can take any number of arguments.
Calling other functions. Functions can call other functions. ← library or user-defined

Gaussian Cumulative Distribution Function

Goal. Compute Gaussian cdf $\Phi(z)$.
Challenge. No "closed form" expression and not in Java library.



Taylor series

$$\begin{aligned}\Phi(z) &= \int_{-\infty}^{z} \phi(x)\,dx \\ &= \frac{1}{2} + \phi(z)\left(z + \frac{z^3}{3} + \frac{z^5}{3\cdot 5} + \frac{z^7}{3\cdot 5\cdot 7} + \ldots\right)\end{aligned}$$

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

Java function for $\Phi(z)$

```
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);
   }
}
```

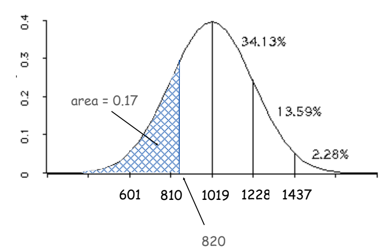
accurate with absolute error less than $8 * 10^{-16}$

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

SAT Scores

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

A. $\Phi(820, \mu, \sigma) \approx 0.17051$. [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 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.

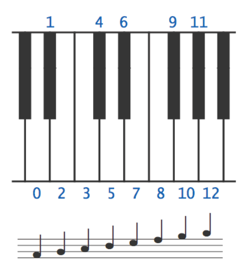
Digital Audio

Crash Course in Sound

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

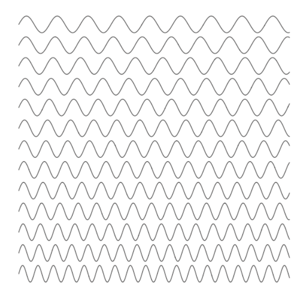
Concert A. Sine wave, scaled to oscillated at 440Hz.

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



note	*i*	*frequency*
A	0	440.00
A♯ *or* B♭	1	466.16
B	2	493.88
C	3	523.25
C♯ *or* D♭	4	554.37
D	5	587.33
D♯ *or* E♭	6	622.25
E	7	659.26
F	8	698.46
F♯ *or* G♭	9	739.99
G	10	783.99
G♯ *or* A♭	11	830.61
A	12	880.00

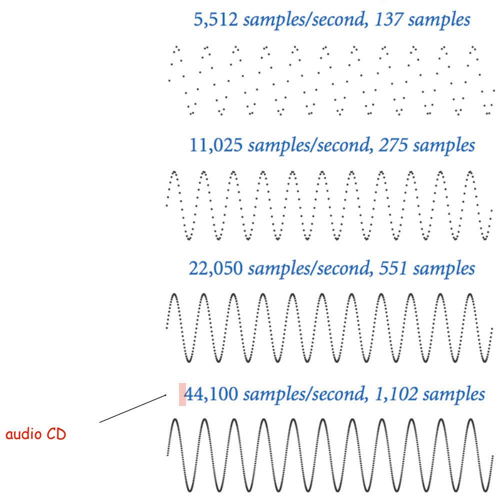
$440 \times 2^{i/12}$



Notes, numbers, and waves

Digital Audio

Sampling. Represent curve by sampling it at regular intervals.



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

Musical Tone Function

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

```
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

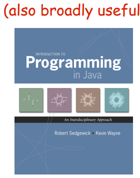
Standard audio. Library for playing digital audio.

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`.

library developed for this course (also broadly useful)

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



Warmup: Musical Tone

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

```
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.

```
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);
      }
   }
}
```

```
% 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
```

TEQ on Functions 2

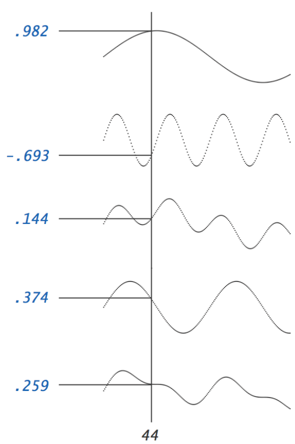
What sound does the following program produce?

```
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.Random();
   return 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




```
lo = tone(220, .0041);
lo[44] = .982

hi = tone(880, .0041);
hi[44] = -.693

h = sum(hi, lo, .5, .5);
h[44] = .5*lo[44]+.5*hi[44];
      = .5*.982 - .5*.693 = .144

A = tone(440, .0041);
A[44] = .374

sum(A, h, .5, .5);
A[44] + h[44] = .5*.144 + .5*.374
              = .259
```


Harmonics

```
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
}
```

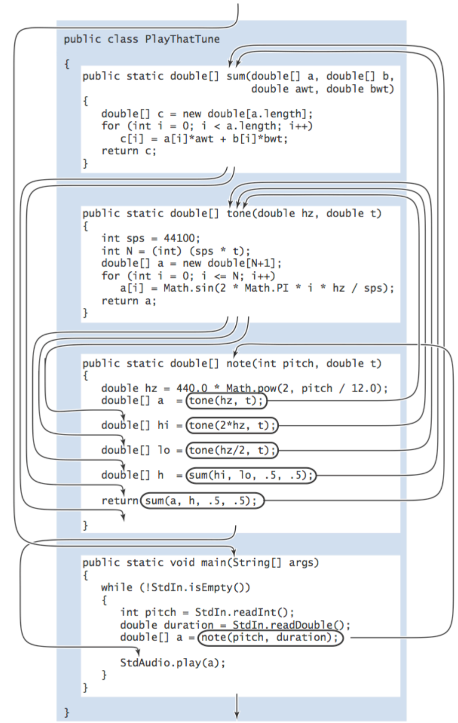
Harmonics

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

```
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
```



2.2 Libraries and Clients

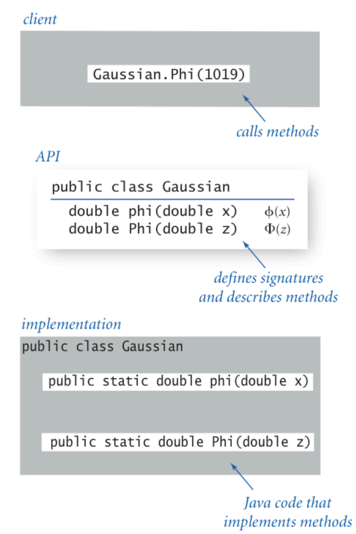
Libraries

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.



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.
}
```


Standard 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.

```
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();
      }
   }
}
```

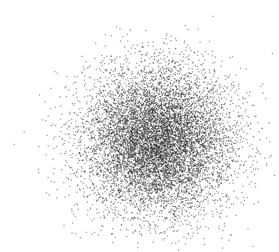
```
% java StdRandom 5
 61 21.76541  true 9.30910
 57 43.64327 false 9.42369
 31 30.86201  true 9.06366
 92 39.59314  true 9.00896
 36 28.27256 false 8.66800
```


Using a Library

```
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);
      }
   }
}
```

use library name to invoke method

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



Standard Statistics

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

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 *sample variance*

Standard Statistics

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

```
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

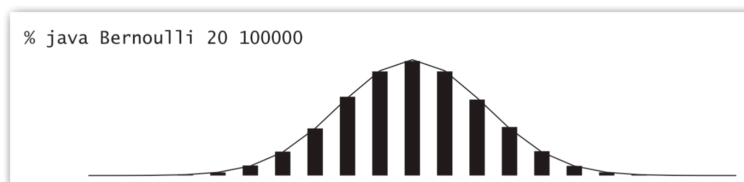
Modular Programming

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.



Bernoulli Trials

```
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;
   }
```

flip N fair coins; return # heads

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

perform T trials of N coin flips each

```
      int[] freq = new int[N+1];
      for (int i = 0; i < T; i++)
         freq[binomial(N)]++;
```

compute frequency of occurrence of each count of heads

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

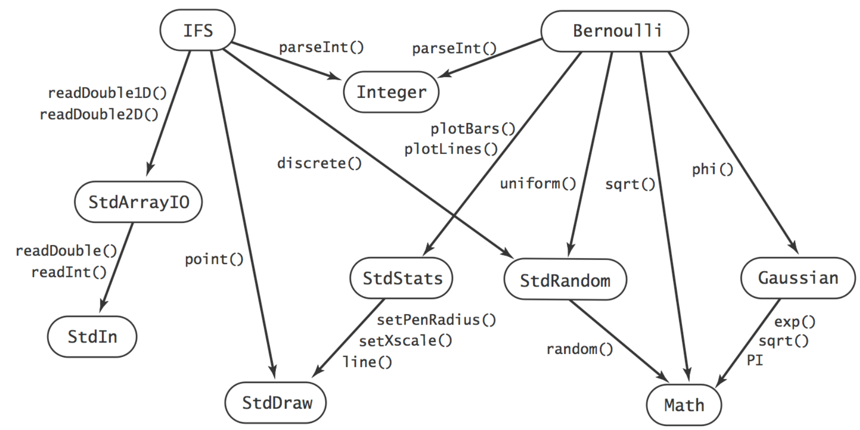
plot histogram of frequencies

```
      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);
   }
}
```

theoretical prediction

Dependency Graph

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.