2.6 Functions

Functions (Static Methods)

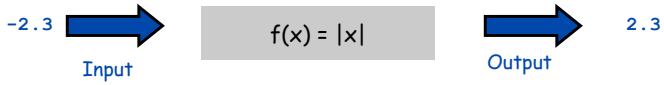
Java function.
- Takes zero or more input arguments.
- Returns one output value.

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.sqrt`, `Integer.parseInt`.
- COS 126 library functions: `StdDraw.spot`, `StdIn.readInt`.
- User-defined functions: `main`.

Mathematical Functions

Scientists use mathematical functions to calculate formulas.
- Use built-in functions when possible.
- Build your own when not available.




```
public static double abs(double x) {
   if (x >= 0.0) return  x;
   else          return -x;
}
```

user-defined function

Build A Library of Mathematical Functions

Java makes it easy to build your own libraries.
- Can substitute `MyMath.sqrt` for `Math.sqrt`.
- Can add functionality to `MyMath` not included in `Math` library.

```
public class MyMath {
   public static double abs(double x) {
      if (x >= 0.0) return  x;
      else          return -x;
   }                                  absolute value

   public static double sqrt(double c) {
      double EPSILON = 1E-15;
      double t = c;
      while (Math.abs(t - c/t) > EPSILON *t)
         t = (c/t + t) / 2.0;
      return t;
   }                                  Newton's method

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

Use the Library in a Program

To use the `MyMath` library.

- Put a copy of `MyMath.java` in current directory.
- Write a client program that uses it.

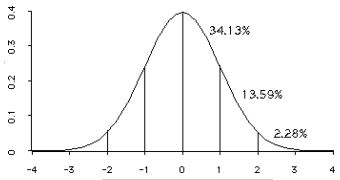
```
public class Die {
   public static void main(String args[]) {
      int die = 1 + MyMath.random(6);
      System.out.println(die);
   }
}
```

```
% javac Die.java

% java Die
5

% java Die
2
```


Gaussian Distribution

Standard Gaussian distribution.

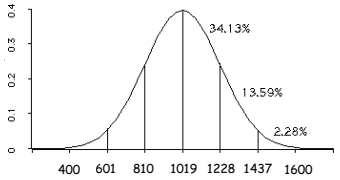
- "Standard normal distribution."
- "Bell curve."
- Basis of most statistical analysis in social and physical sciences.



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

Ex: 2000 SAT scores.

- Follows a Gaussian distributed with:
  - mean μ = 1019
  - standard deviation σ = 209
- 2.275% get above 1437.

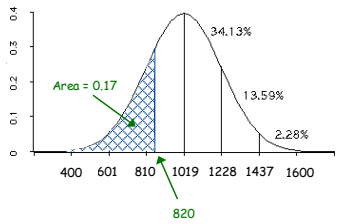


$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

SAT Scores

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

- $\Phi(820, \mu, \sigma)$ = 0.17051....
- Approximately 17%.



Challenge: write a Java program to compute this.

Gaussian Distribution

Why relevant in the sciences?

- Models a wide range of natural phenomena and random processes.
  - weights of humans, heights of trees in a forest
  - exam scores, investment returns

Why relevant in mathematics?

- Central limit theorem: under very general conditions, average of a set of variables tends to the Gaussian distribution.

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*

SAT Client

Client program is easy to write, given desired library function.

```
public class SAT {
   public static void main(String args[]) {
      double mu       = 1019;
      double sigma    =  209;
      double z        =  820;
      double fraction = MyMath.Phi(z, mu, sigma);
      System.out.println(fraction);
   }
}
```

```
% java SAT
0.17050967431793962
```

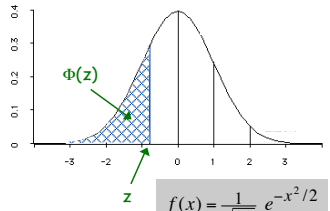
Next: implement `MyMath.Phi`.

Gaussian Distribution

Compute Gaussian cumulative distribution function.

- No "closed form" expression and not in Java library.
- Express in terms of error function.

$$\Phi(z) = \int_{-\infty}^{z} f(x)\,dx = \frac{1 + \mathrm{erf}(z/\sqrt{2})}{2}$$



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

Error function.

- If z ≥ 0,

$$\begin{aligned} erf(z) &= \int_0^z \frac{2}{\sqrt{\pi}} e^{-x^2}\,dx \\ &\approx 1 - te^{(-z^2 - 1.26551223 + 1.00002368t + 0.37409196t^2 + 0.09678418t^3 - 0.18628806t^4 \\ &\qquad + 0.27886807t^5 - 1.13520398t^6 + 1.48851587t^7 - 0.82215223t^8 + 0.17087277t^9)} \end{aligned}$$

where $t = \frac{1}{1+z/2}$

- Else erf(z) = -erf(-z).

Fractional error less than than 1.2×10^{-7}.
Reference: Numerical Recipes 6.2

Building Layers of Abstraction

Functions enable you to build a new layer of abstraction.

- Takes you beyond pre-packaged libraries.
- You build the tools you need.
  - Ex: erf and Φ

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.

Computing erf(z) and Φ(z) in Java

```
public static double erf(double z) {
   double t = 1.0 / (1.0 + 0.5 * Math.abs(z));
   double ans = 1 - t * Math.exp( -z*z - 1.26551223 +
                                  t * ( 1.00002368 +
                                  t * ( 0.37409196 +
                                  t * ( 0.09678418 +
                                  t * (-0.18628806 +
                                  t * ( 0.27886807 +
                                  t * (-1.13520398 +
                                  t * ( 1.48851587 +
                                  t * (-0.82215223 +
                                  t * ( 0.17087277)))))))));
   if (z >= 0) return  ans;
   else        return -ans;
}
```

Horner's method

$$c_3 x^3 + c_2 x^2 + c_1 x^1 + c_0$$
$$=$$
$$c_0 + x(c_1 + x(c_2 + x(c_3)))$$

```
public static double Phi(double z) {
   return 0.5 * (1.0 + erf(z / (Math.sqrt(2.0))));
}
```

overloaded methods

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

Black-Scholes

Black-Scholes option pricing model.

- Option = right to buy stock at some future date for fixed price.
- Model stock price with stochastic differential equation.
- Ex: how much is a given option worth?
- Won 1997 Nobel Prize in Economics.

Model assumptions.

- Stock price follows geometric Brownian motion.
- Risk free interest rate is constant and known.
- No dividends.
- Markets are efficient.
- No commissions.
- No arbitrage.

Black-Scholes Formula

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial^2 S} = rf$$

$$\begin{aligned} f(0) &= S\,\Phi(d_1) - Xe^{-rT}\Phi(d_2) \\ d_1 &= \frac{\ln(S/X) + (r + \frac{1}{2}\sigma^2)T}{s\sqrt{T}} \\ d_2 &= d_1 - s\sqrt{T} \end{aligned}$$

Parameter	Description
f(t)	value of a call option at time t
S	current price of stock
X	strike price
r	risk free interest rate
σ	standard deviation of stock return (volatility)
T	time until option expires

Black-Scholes Formula in Java

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial^2 S} = rf$$

$$\begin{aligned} f(0) &= S\,\Phi(d_1) - Xe^{-rT}\Phi(d_2) \\ d_1 &= \frac{\ln(S/X) + (r + \frac{1}{2}\sigma^2)T}{s\sqrt{T}} \\ d_2 &= d_1 - s\sqrt{T} \end{aligned}$$

```
public class BlackScholes {
   static double call(double S, double X, double r, double sigma, double T) {
      double temp = sigma * Math.sqrt(T);
      double d1   = (Math.log(S/X) + (r + 0.5*sigma*sigma) * T) / temp;
      double d2   = d1 - temp;
      return S * MyMath.Phi(d1) - X * Math.exp(-r * T) * MyMath.Phi(d2);
   }
   public static void main(String args[]) {
      double S     = Double.parseDouble(args[0]);
      double X     = Double.parseDouble(args[1]);
      double r     = Double.parseDouble(args[2]);
      double sigma = Double.parseDouble(args[3]);
      double T     = Double.parseDouble(args[4]);
      System.out.println(call(S, X, r, sigma, T));
   }
}
```

invokes library function

Black-Scholes

Does Black-Scholes accurately model option price?

Parameter	Microsoft (June 9, 2003)	General Electric (June 9, 2003)
S	$23.75	$30.14
X	$15.00	$15.00
r	1%	1%
σ	35.0%	33.2%
T	0.5 years	0.25 years

⇐ historical estimate

```
% java BlackScholes 23.75 15.0 0.01 0.350 0.50
8.879159279691955

% java BlackScholes 30.14 15.0 0.01 0.332 0.25
15.177462481562186
```

actual option price = $9.10

actual option price = $14.50

Functions (Static Methods)

Java function.
- Takes zero or more input arguments.
- Returns one output value.

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

Modular Programming

Programmers use functions to build modular programs.
- Divide program into self-contained pieces.
- Test each piece individually.
- Combine pieces to make program.

Ex: gambler's ruin.
- Method 1: put everything in `main`.
- Method 2: break up program into self-contained pieces.
  - flip a coin
  - play the game once
  - repeat the game and tabulate statistics

Gambler's Ruin Revisited

```
public class Gambler {
   public static boolean flip() {
      return (Math.random() < 0.5);
   }                                  flip fair coin

   public static boolean winsGamble(int stake, int goal) {
      while ((stake > 0) && (stake < goal))
         if (flip()) stake++;
         else        stake--;
      return (stake == goal);
   }                                          play the game once

   public static void main(String[] args) {
      int stake = Integer.parseInt(args[0]);
      int goal  = Integer.parseInt(args[1]);
      int N     = Integer.parseInt(args[2]);
      int wins  = 0;
      for (int i = 0; i < N; i++)
         if (winsGamble(stake, goal)) wins++;
      System.out.println(wins + " wins of " + N);
   }
}
```


Craps

What is probability of winning a "pass bet" in craps?
- Roll two dice, and let x be sum.
- If x is 7 or 11, you win instantly.
- Else if x is 2, 3, or 12, you lose instantly.
- Otherwise repeatedly roll two dice until sum is 7 or x.
  - if sum is x, you win
  - if sum is 7 you lose

```
public class Craps {
   . . .
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int wins = 0;
      for (int i = 0; i < N; i++)
         if (winsPassBet()) wins++;
      System.out.println("Win % = " + 1.0 * wins / N);
   }
}
```

Monte Carlo simulation

Craps: Helper Functions

```
public static int sumOfTwoDice(int sides) {
   int x = 1 + MyMath.random(sides);
   int y = 1 + MyMath.random(sides);
   return x + y;
}
```

```
public static boolean winsPassBet() {
   int x = sumOfTwoDice(6);
   if (x == 7 || x == 11)           return true;
   if (x == 2 || x == 3 || x == 12) return false;
   while (true) {
      int y = sumOfTwoDice(6);
      if (y == 7) return false;
      if (y == x) return true;
   }
}
```

```
% java Craps 1000000
Win % = 0.493396

% java Craps 1000000
Win % = 0.492537
```

Exact answer = 244/495.

Numerically Solving an Initial Value ODE

Lorenz attractor.

- Idealized atmospheric model to describe turbulent flow.
- Convective rolls: warm fluid at bottom, rises to top, cools off, and falls down.
- Rayleigh-Bernard cell.

$$\frac{dx}{dt} = -\sigma x + \sigma y$$
$$\frac{dy}{dt} = -xz + rx - y$$
$$\frac{dz}{dt} = xy - bz$$

x = fluid flow velocity

y = temperature difference between ascending and descending currents

z = distortion of vertical temperature profile from linearity

σ= 10 = Prandtl number
b = 8 / 3 = width-to-height ratio of convective layer
r = 28 = temperature difference between top and bottom

Euler's Method

Euler's method: to numerically solve initial value ODE.

- Choose Δt sufficiently small.
- Approximate function at time t by tangent line at t.
- Estimate value of function at time t + Δt according to tangent line.
- Increment time to t + Δt.
- Repeat.

$$x_{t+\Delta t} = x_t + \Delta t \frac{dx}{dt}(x_t, y_t, z_t)$$
$$y_{t+\Delta t} = y_t + \Delta t \frac{dy}{dt}(x_t, y_t, z_t)$$
$$z_{t+\Delta t} = z_t + \Delta t \frac{dz}{dt}(x_t, y_t, z_t)$$

Advanced methods: use less computation to achieve desired accuracy.

- 4th order Runge-Kutta: evaluate slope four times per step.
- Variable time step: automatically adjust timescale Δt.
- See COS 323.

Lorenz Attractor: Java Implementation

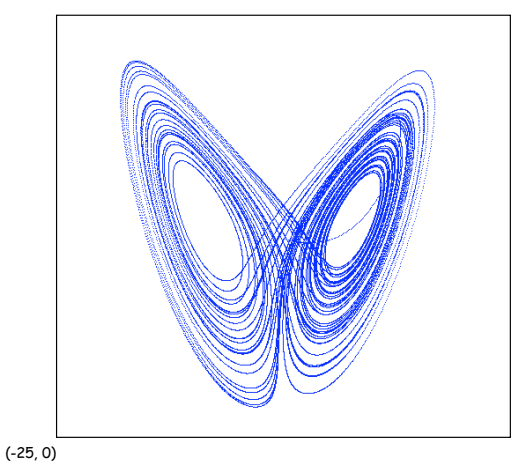
```
public class Lorenz {
   public static double dx(double x, double y, double z)
      { return -10*(x - y);      }
   public static double dy(double x, double y, double z)
      { return -x*z + 28*x - y; }
   public static double dz(double x, double y, double z)
      { return x*y - 8*z/3;      }

   public static void main(String[] args) {
      double x = 0.0, y = 20.0, z = 25.0;
      double dt = 0.001;
      StdDraw.create(512, 512);
      StdDraw.setScale(-25, 0, 25, 50);
      while (true) {
         double xnew = x + dt * dx(x, y, z);
         double ynew = y + dt * dy(x, y, z);
         double znew = z + dt * dz(x, y, z);
         x = xnew; y = ynew; z = znew;
         StdDraw.go(x, z);
         StdDraw.spot();
         StdDraw.pause(10);
      }
   }
}
```

Euler's method

plot x vs. z

The Lorenz Attractor

```
% java Lorenz
```



Butterfly Effect

Experiment.

- Initialize y = 20.01 instead of y = 20.
- Plot original trajectory in blue, perturbed one in magenta.
- What happens?

Chaos.

- Sensitive dependence on initial conditions.
- Unpredictability of aperiodic systems like the weather.
- Lorenz attractor never repeats itself, but produces orderly pattern.
- Property of system, not of numerical solution approach.

"Predictability: Does the Flap of a Butterfly's Wings in Brazil set off a Tornado in Texas?" - *Title of 1972 talk by Edward Lorenz*

Functions

Why use functions?

- Makes code easier to understand.
- Makes code easier to debug.
- Makes code easier to maintain.
- Makes code easier to re-use.