2.1 Functions

Java functions. Easy to write your own.

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.abs(), Integer.parseInt().
- User-defined functions: main().

Flow of Control

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

Java Functions

Flow of control. Functions provide a new way to control the flow of execution of a program.

public class Newton {
  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;
  }
}

% java Newton 1 2 3
1.0
1.414213562373095
1.7320508075688772
Scope

The scope of a variable is the set of statements that can refer to that name.
- Scope of a variable defined within a block is limited to the statements in that block.
- Best practice: declare variables to limit their scope.

```java
public class Scope {
    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++)
            System.out.println(i + " " + cube(i));
    }
}
```

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 Gaussian.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 Distribution

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

Ex. 2000 SAT scores follows a Gaussian distributed with mean $\mu = 1019$, stddev $\sigma = 209$.

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

```java
public class Gaussian {
    // as before
    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 += 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);
    }
}
```

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

**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

**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%]

**Building Libraries**
To use the standard random library:
- Put a copy of `StdRandom.java` in current directory.
- Write a client program that uses it.

```java
public class StdRandom {
    public static double uniform(double a, double b) {
        return a + Math.random() * (b-a);
    }
    public static int uniform(int N) {
        return (int) (Math.random() * N);
    }
    public static boolean bernoulli(double p) {
        return Math.random() < p;
    }
    public static double gaussian() {
        // recall Assignment 0
    }
    public static double discrete(double[] p) {
        // next slide
    }
}
```

Example Library to generate pseudo-random numbers.

Using a Library

Discrete Distribution

Discrete distribution. Given an array of weights (that sum to 1), choose an index at random with probability equal to its weight.

```
public static int discrete(double[] p) {
    // check that weights are nonnegative and sum to 1
    double r = Math.random();
    double sum = 0.0;
    for (int i = 0; i < p.length; i++) {
        sum += p[i];
        if (sum >= r) return i;
    }
    return -1; // something went wrong
}
```

Building Libraries

Functions enable you to build a new layer of abstraction.
- Takes you beyond pre-packaged libraries.
- You build the tools you need: `Math.Phi()`, `StdRandom.uniform()`, ...

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.

```bash
1. javac LoadedDie.java
2. java LoadedDie 10
   6 5 1 2 6 6 2 6 6 6
```
Modular Programming

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

Ex. Coupon collector.
Read parameter $N$.
Choose a random card between 0 and $N-1$.
Run one coupon collector simulation.
Repeat simulation many times.
Tabulate statistics.
Print results.

Coupon Collector

Coupon collector function. Given $N$ coupon types, how many coupons do you need to collect until you have at least one of each type?

```java
public class Coupon {
    public static int collect(int N) {
        boolean[] found = new boolean[N];
        int cardcnt = 0, valcnt = 0;
        while (valcnt < N) {
            int r = StdRandom.uniform(N);
            if (!found[r]) valcnt++;
            found[r] = true;
            cardcnt++;
        }
        return cardcnt;
    }
}
```
Computational experiment.
- For each $N$, collect coupons until at least one of each type.
- Repeat experiment several times for each value of $N$.
- Tabulate statistics and analyze results.

```java
public class CouponExperiment {
    public static void main(String[] args) {
        int TRIALS = Integer.parseInt(args[0]);
        double[] results = new double[TRIALS];
        for (int N = 100; N <= 10000; N *= 10) {
            for (int i = 0; i < TRIALS; i++)
                results[i] = Coupon.collect(N);
            double mean = StdStats.mean(results);
            double stddev = StdStats.stddev(results);
            System.out.printf("%8d%8.2f%8.2f
", N, mean, stddev);
        }
    }
}
```

<table>
<thead>
<tr>
<th>TRIALS</th>
<th>100</th>
<th>516.97</th>
<th>126.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>7511.94</td>
<td>1300.84</td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>97778.80</td>
<td>12795.39</td>
<td></td>
</tr>
</tbody>
</table>

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.

Modular programming. Build relatively complicated program by combining several small, independent, modules.