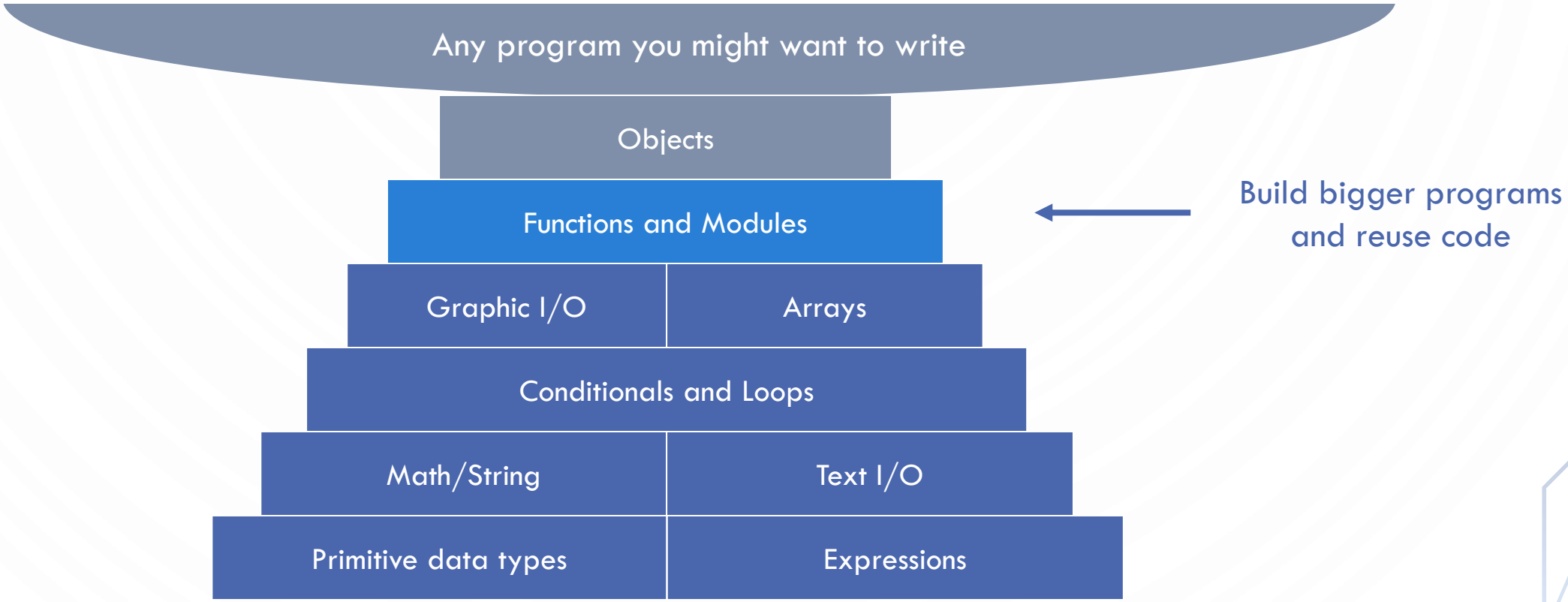
CMSC 150
INTRODUCTION TO COMPUTING

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING IN JAVA: AN INTERDISCIPLINARY APPROACH, SEDGEWICK AND WAYNE (PEARSON ADDISON-WESLEY 2007)

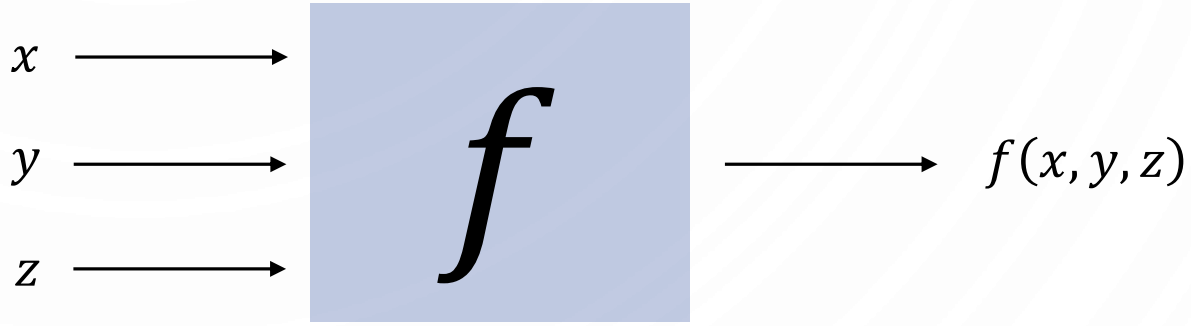
LECTURE 5

- FUNCTIONS

A FOUNDATION FOR PROGRAMMING



2.1 FUNCTIONS



FUNCTIONS
CALLED STATIC METHODS IN JAVA

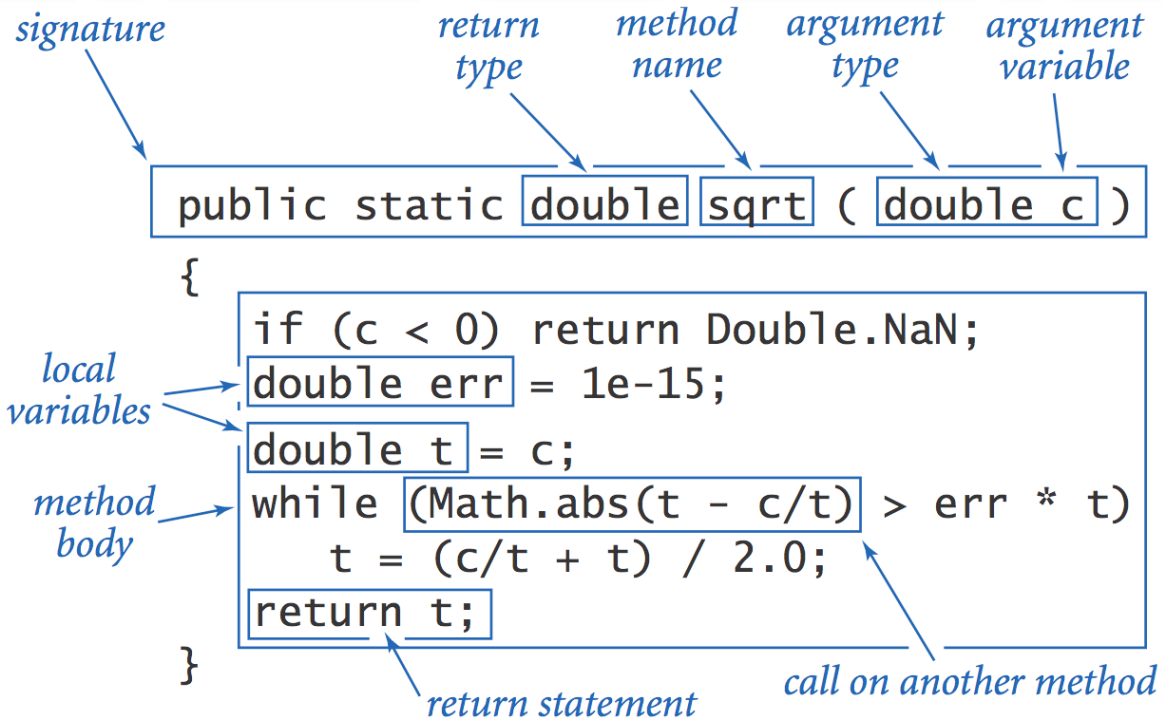
- Java function.
  - Takes zero or more input arguments.
  - Returns one output value.
  - Side effects (e.g., output to standard draw).
- Applications.
  - Scientists use mathematical functions to calculate formulas.
  - Programmers use functions to build modular programs.
  - You use functions for both.

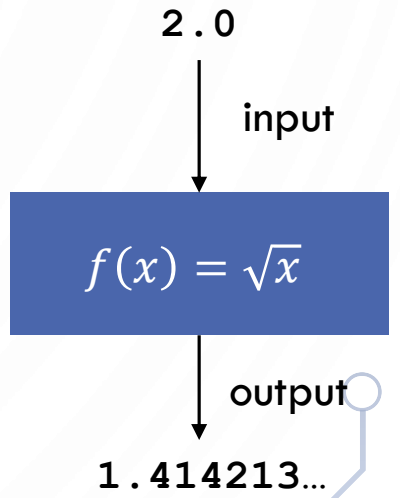
- Examples.
  - Built-in functions: Math.random(), Math.abs(), Integer.parseInt().
  - Our I/O libraries: StdIn.readInt(), StdDraw.line(), StdAudio.play().
  - User-defined functions: main().

Java functions are More general than mathematical functions!!

ANATOMY OF A JAVA FUNCTION

- Java functions. Easy to write your own.





EXAMPLE

- Lets write a function to compute a random integer between $[a, b]$
- Lets write a function to compute the minimum of an array of double
- Lets write a function to output a multiarray of long

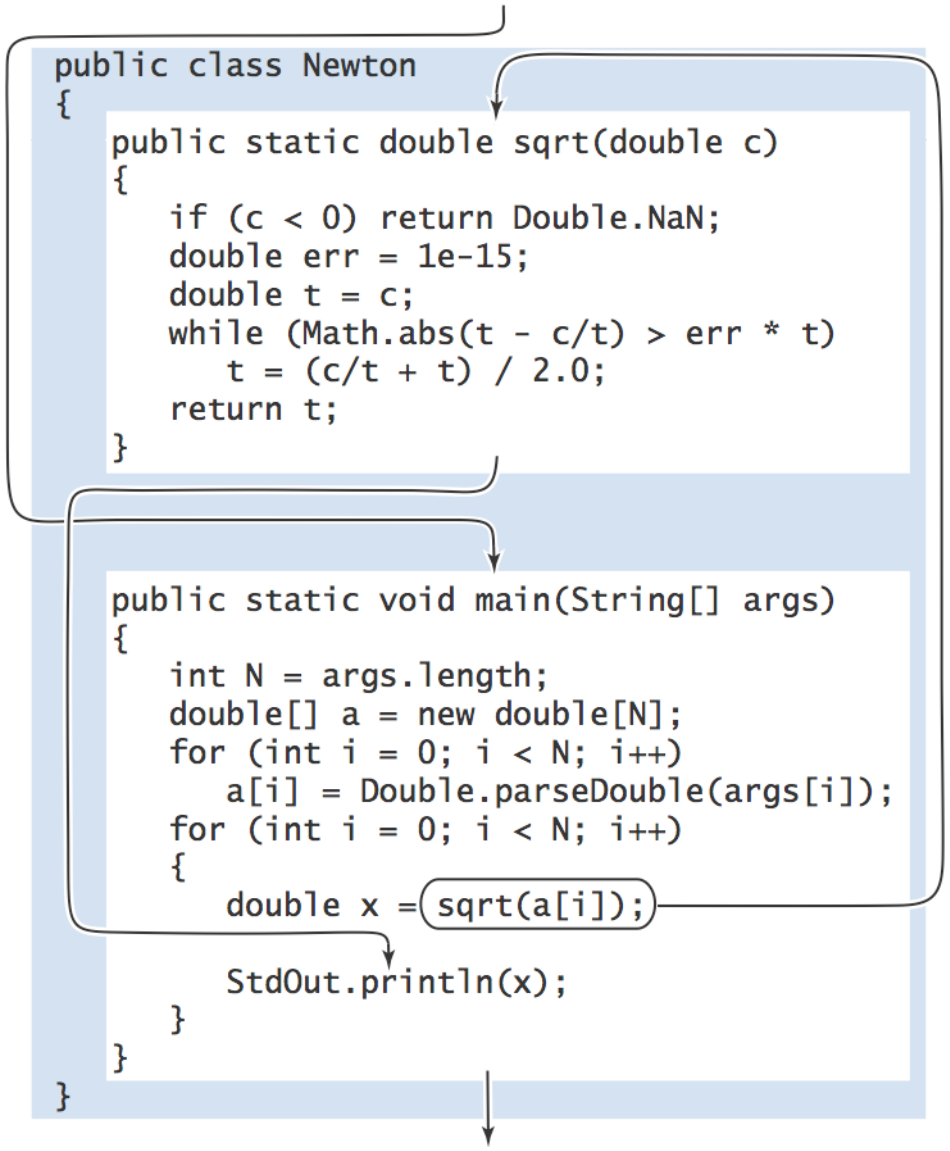
EXERCISE – WITH A PARTNER

- Write a function to determine if two circles collide overlap
- Write a function that randomly shuffles an array of float

FLOW OF CONTROL

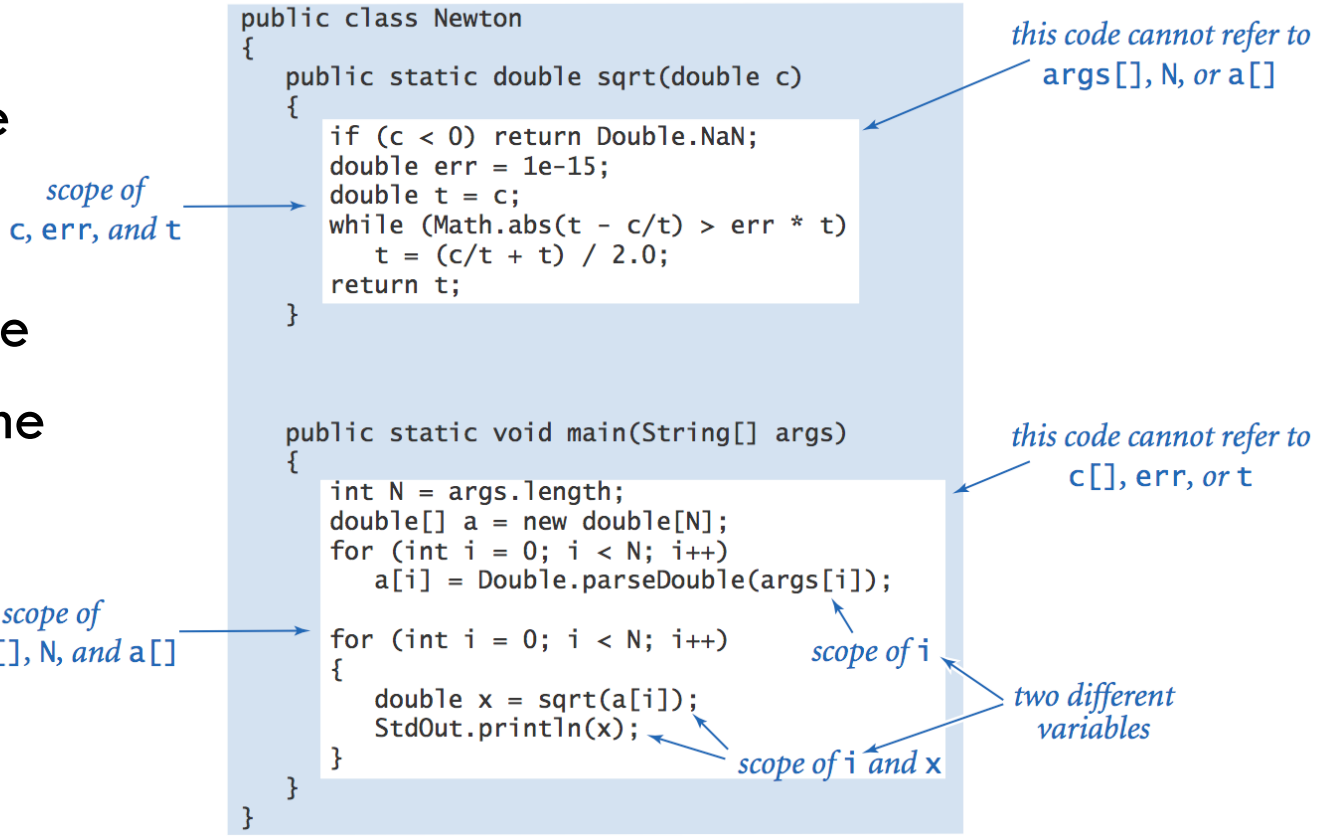
- Key point. Functions provide a new way to control the flow of execution.
- What happens when a function is called:
  - Control transfers to the function code.
  - Argument variables are assigned the values given in the call.
  - Function code is executed.
  - Return value is assigned in place of the function name in calling code.
  - Control transfers back to the calling code.

implicit return statement at end of void function



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.
- Best practice: declare variables to limit their scope.



FUNCTION EXAMPLES

absolute value of an `int` *value*

```
public static int abs(int x)
{
   if (x < 0) return -x;
   else       return  x;
}
```

absolute value of a `double` *value*

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

overloading

primality test

```
public static boolean isPrime(int N)
{
   if (N < 2) return false;
   for (int i = 2; i <= N/i; i++)
      if (N % i == 0) return false;
   return true;
}
```

multiple arguments

hypotenuse of a right triangle

```
public static double hypotenuse(double a, double b)
{  return Math.sqrt(a*a + b*b);  }
```

OVERLOADING

- Having two methods with the same, but different signatures (parameters + return type) is referred to as overloading. Example:

```
public static void print(int i);
public static void print(long l);
public static void print(double d);
public static int UniformRandom(int a, int b);
public static double UniformRandom(double a, double b)
public static char UniformRandom();
```

MEMORY AND FUNCTIONS
PASSING-BY-VALUE

- On parameters
  - Values are always “passed-by-value”
  - Jargon term for values of variables being copied into the function

"You're exactly the kind of applicant we're looking for."

```
public static void foo(int i) {
    i++; //the function's i is modified, not the variable passed to the function
    System.out.println(i); //Prints 6!
}
public static void bar() {
    int i = 5;
    foo(i);
    System.out.println(i); //Prints 5!
}
```

MEMORY AND FUNCTIONS
PASSING-BY-VALUE

- Arrays
  - An array's value is the length and the memory address of its elements
  - Only this information is copied, NOT the elements

```
public static char[] foo(char[] c) {
    c = new char[20]; //create new array
    return c; //return only the new memory location + length
}
public static void bar() {
    char[] c = new char[10];
    char[] d = foo(c); //c is still length 10, d is new array
}
```

FUNCTION CHALLENGE 1A

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

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

```
1.  public class Cubes1 {
2.    public static int cube(int i) {
3.      int j = i * i * i;
4.      return j;
5.    }
6.    public static void main(String[] args) {
7.      int N = Integer.parseInt(args[0]);
8.      for (int i = 1; i <= N; i++)
9.        StdOut.println(i + " " + cube(i));
10.   }
11. }
```

FUNCTION CHALLENGE 1B

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

Compile error!

```
1.  public class Cubes2 {
2.    public static int cube(int i) {
3.      int i = i * i * i;
4.      return i;
5.    }
6.    public static void main(String[] args) {
7.      int N = Integer.parseInt(args[0]);
8.      for (int i = 1; i <= N; i++)
9.        StdOut.println(i + " " + cube(i));
10.   }
11. }
```

FUNCTION CHALLENGE 1C

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

Compile error!

```
1.  public class Cubes3 {
2.    public static int cube(int i) {
3.      i = i * i * i;
4.    }
5.    public static void main(String[] args) {
6.      int N = Integer.parseInt(args[0]);
7.      for (int i = 1; i <= N; i++)
8.        StdOut.println(i + " " + cube(i));
9.    }
10. }
```

FUNCTION CHALLENGE 1D

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

```
% java Cubes4 6
1 1
2 8
3 27
4 64
5 125
6 216
```

```
1.  public class Cubes4 {
2.    public static int cube(int i) {
3.      i = i * i * i;
4.      return i;
5.    }
6.    public static void main(String[] args) {
7.      int N = Integer.parseInt(args[0]);
8.      for (int i = 1; i <= N; i++)
9.        StdOut.println(i + " " + cube(i));
10.   }
11. }
```

FUNCTION CHALLENGE 1E

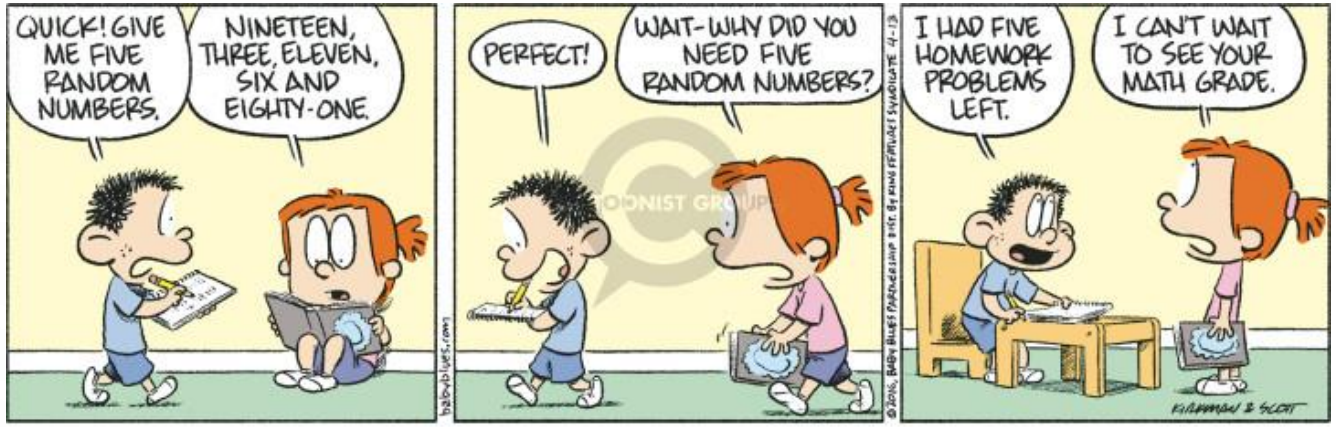
- Q. What happens when you compile and run the following code?

```
% java Cubes5 6
1 1
2 8
3 27
4 64
5 125
6 216
```

```
1.  public class Cubes5 {
2.    public static int cube(int i) {
3.      return i * i * i;
4.    }
5.    public static void main(String[] args) {
6.      int N = Integer.parseInt(args[0]);
7.      for (int i = 1; i <= N; i++)
8.        StdOut.println(i + " " + cube(i));
9.    }
10. }
```

BUILDING FUNCTIONS

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



MORE PROBLEMS

- Write a program to read a list of values from a file. These will be x values.
- Write two functions to plot $f(x) = 3x^4 + \sqrt{x} - 23$
  - One function will compute the above equation for a single x value
  - The other function will evaluate the equation (using the prior function) over all of the x-values
  - The main function will be responsible for reading the data and calling the plot function