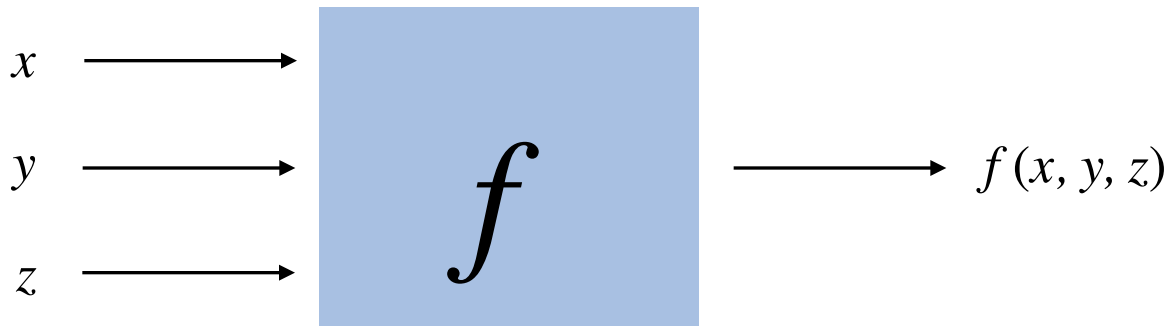
2.1 Functions



Functions (Static Methods)

Java function.

- Takes zero or more input arguments.
- Returns one output value.
- Side effects (e.g., output to standard draw).

more general than mathematical functions

Applications.

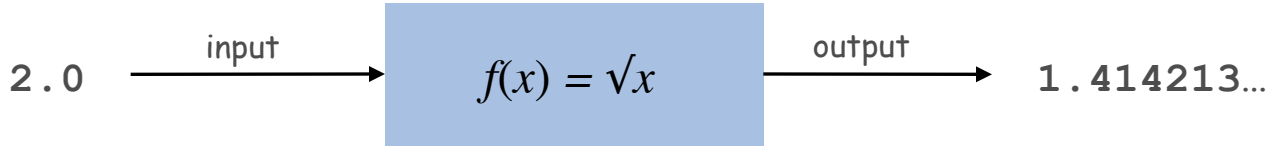
- Scientists use mathematical functions to calculate formulas.
- Programmers use functions to build modular programs.
- You use functions for both.

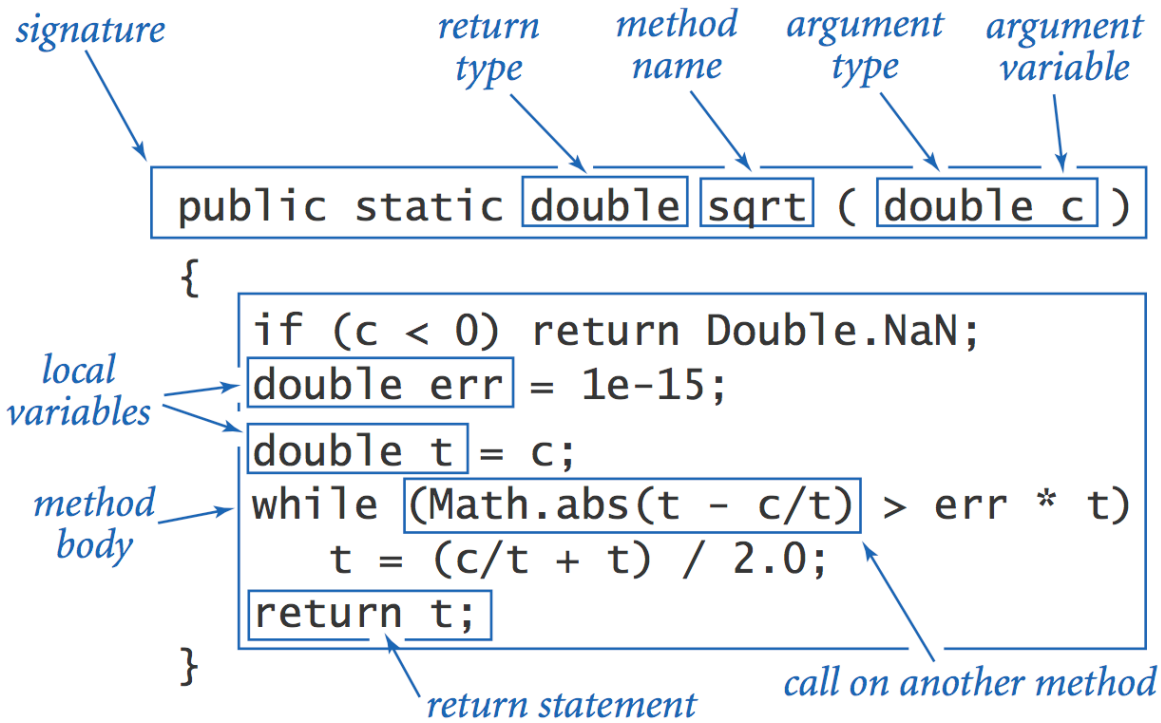
Examples.

- Built-in functions: `Math.random()`, `Math.abs()`, `Integer.parseInt()`.
- Our I/O libraries: `.nextInt()`, `System.out.print()`.
- 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)
   {
      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);
      }
   }
}
```

implicit return statement
at end of void function

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.

```
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);
      }
   }
}
```

Note. This is known as "pass by value."

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.

```
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);
      }
   }
}
```

this code cannot refer to `args[]`, `N`, *or* `a[]`

scope of `c`, `err`, *and* `t`

this code cannot refer to `c[]`, `err`, *or* `t`

scope of `i`

scope of `args[]`, `N`, *and* `a[]`

two different variables

scope of `i` *and* `x`

Best practice: declare variables to limit their scope.

Function Challenge 1a

Q. 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));
   }
}
```

```
% javac Cubes1.java
% java Cubes1 6
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?

```
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++)
         System.out.println(i + " " + cube(i));
   }
}
```

Function Challenge 1c

Q. 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));
   }
}
```

Function Challenge 1d

Q. 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));
   }
}
```

Function Challenge 1e

Q. 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));
   }
}
```

Extra Slides

Function Examples

absolute value of an `int` *value*

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

overloading

absolute value of a `double` *value*

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

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);  }
```