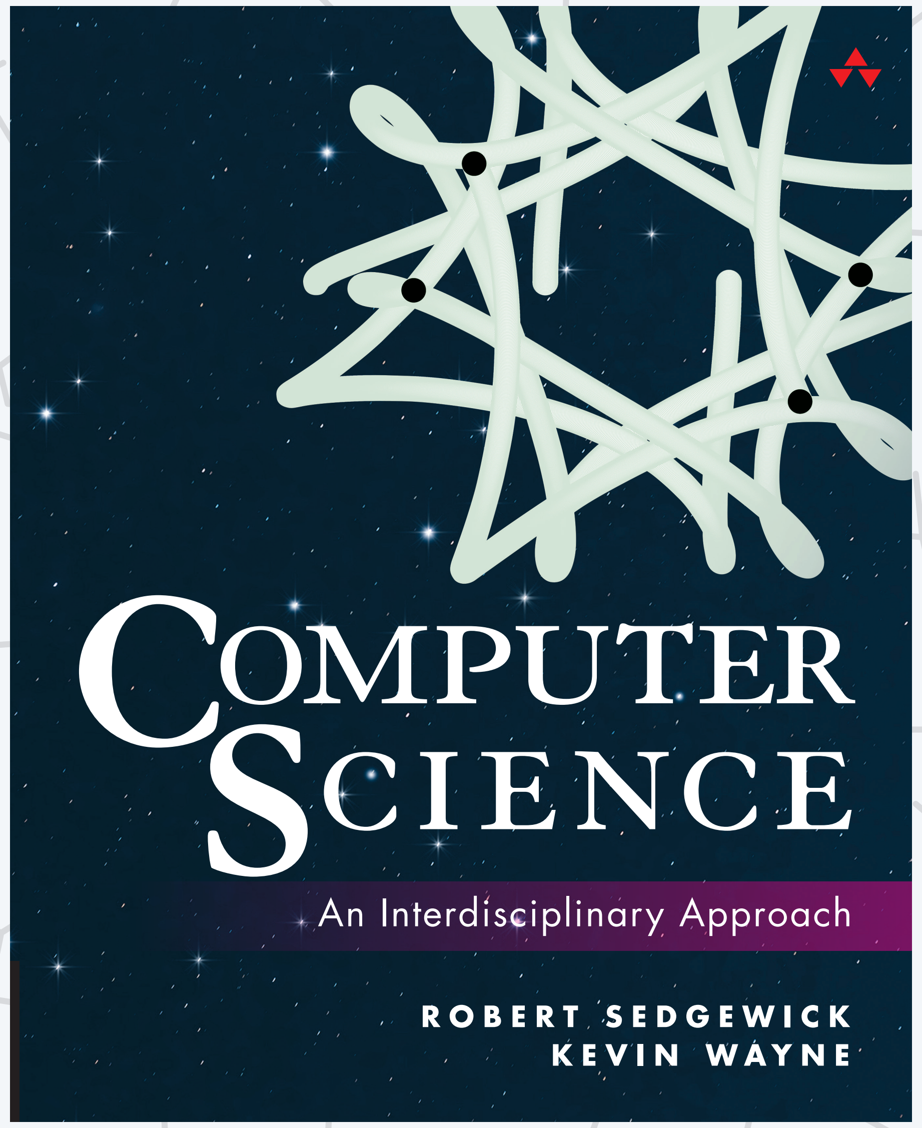
# Computer Science

ROBERT SEDGEWICK | KEVIN WAYNE



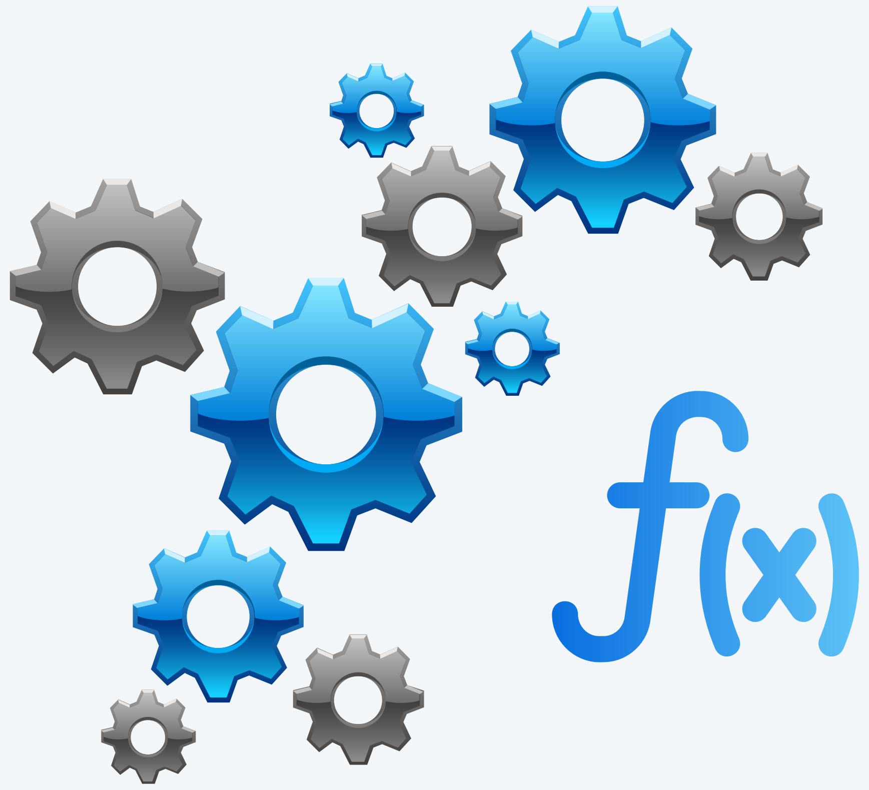
https://introcs.cs.princeton.edu

## 2.1 FUNCTIONS

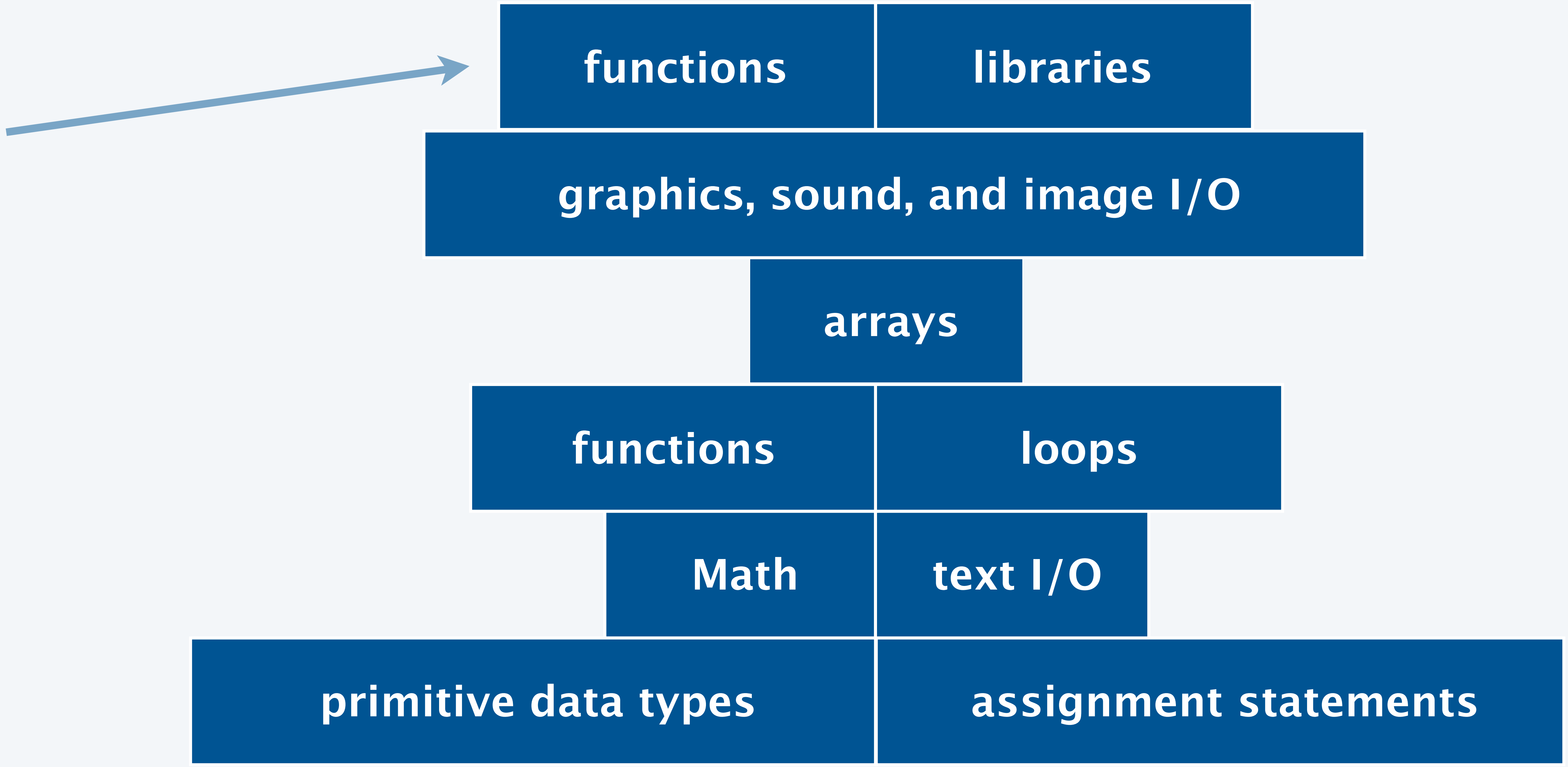
- call by value
- recursion
- what next?

Last updated on 8/6/24 11:50PM

# Basic building blocks for programming



*divide a program into functions*



functions | libraries

graphics, sound, and image I/O

arrays

functions | loops

Math | text I/O

primitive data types | assignment statements

## Summary

Functions. Provide a fundamental way to change flow of control of program.

- Java evaluates the arguments and passes by value to function.
- Function initializes parameter variables with corresponding argument values.
- Function computes a single return value and returns it to caller.

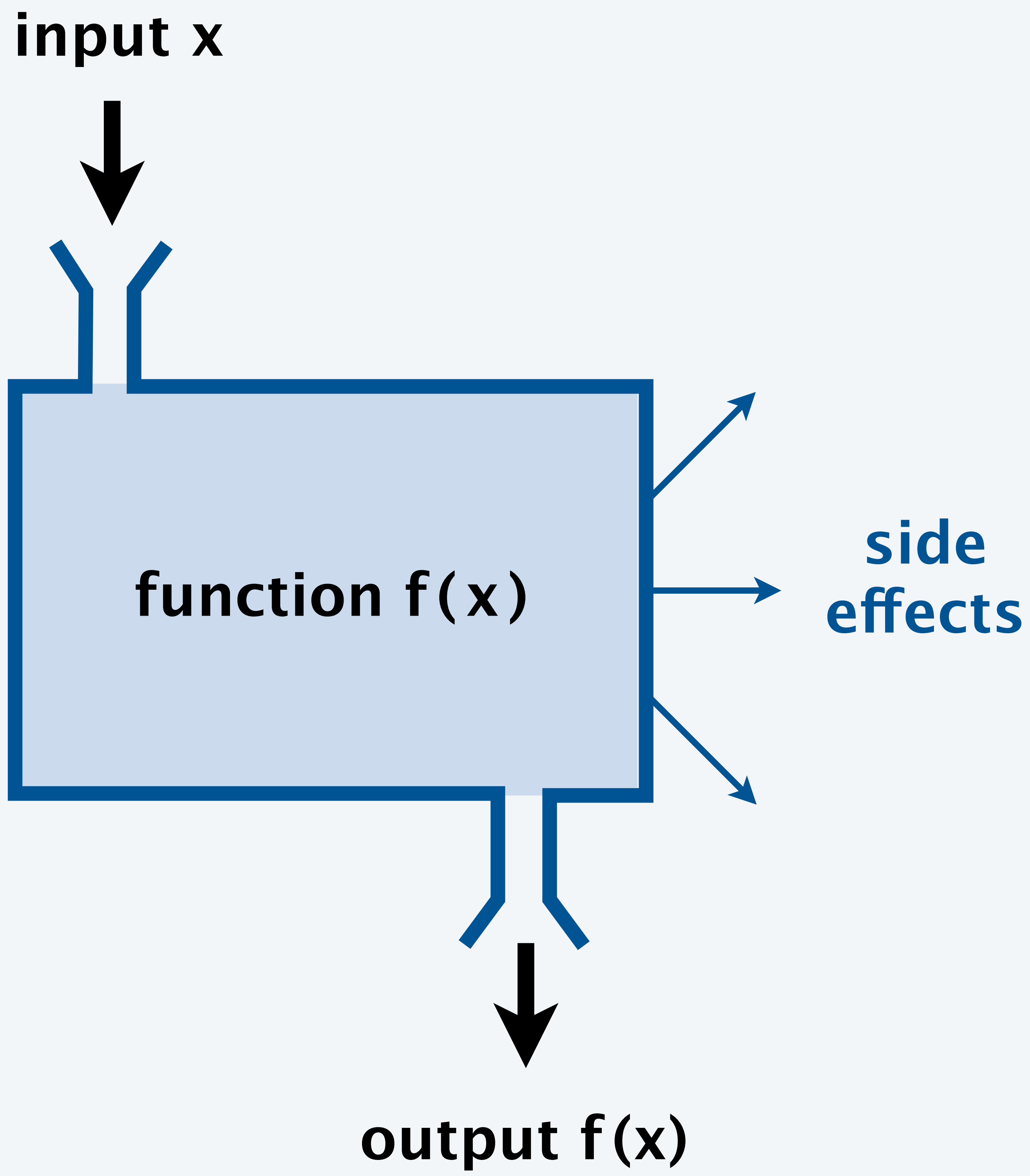
Applications.

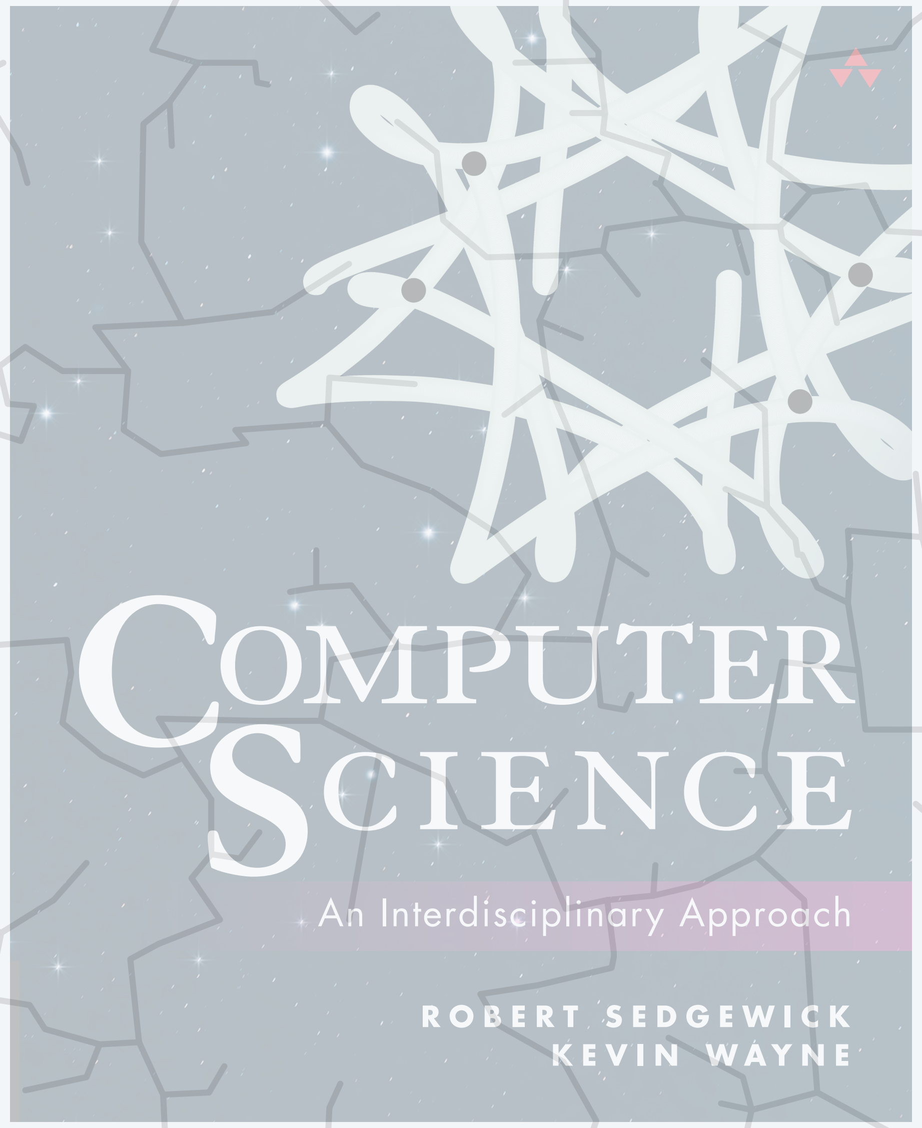
- Scientists use mathematical functions to calculate formulas.
- Programmers use functions to build modular programs.
- You use functions for both.

Last lecture. Write your own functions.

Last precept. Build reusable libraries of functions.

This lecture. How Java passes arguments, and self-referential functions.





https://introcs.cs.princeton.edu

# 2.1 Functions

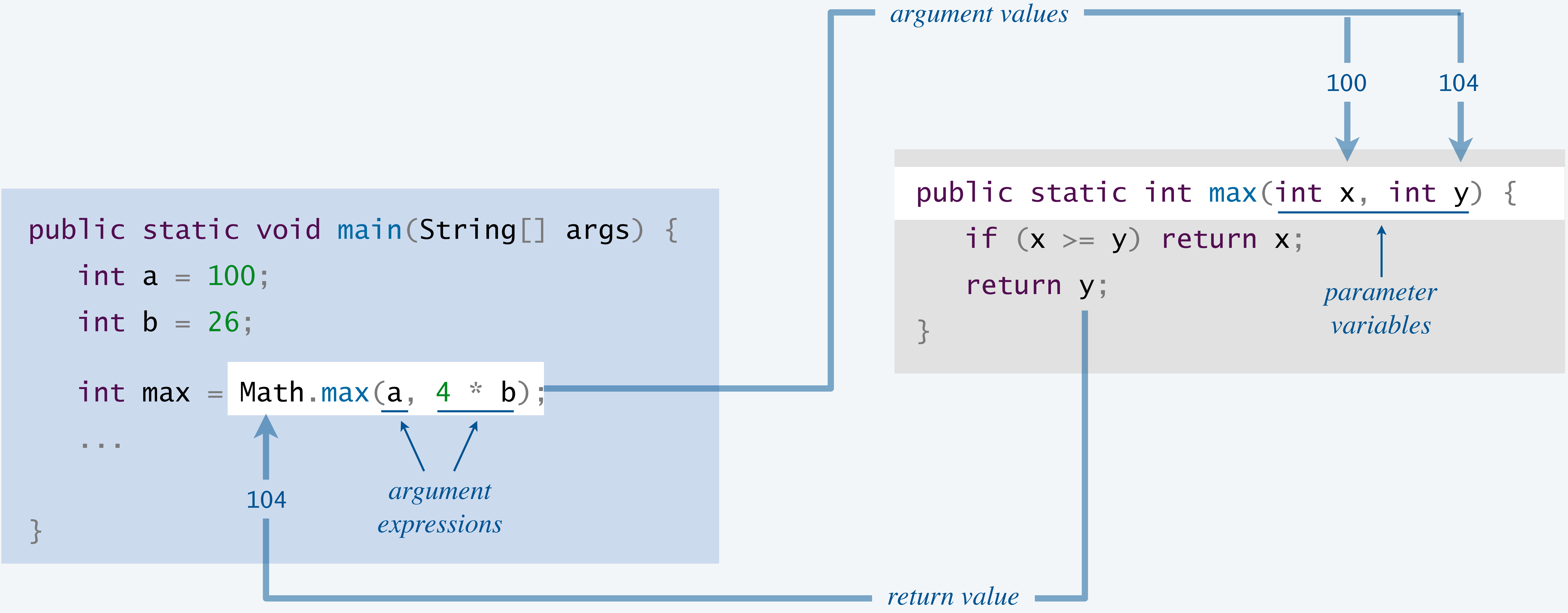
- ▸ *call by value*
- ▸ *recursion*
- ▸ *what next?*

# Call by value

Java uses call by value to pass arguments to methods.

- Java evaluates each argument expression to produce a value. ←
- Java assigns each value to the corresponding parameter variable.

*for primitive types, the value is the data-type value;
for arrays (and other non-primitive types),
the value is an "object reference"*



```
public static void main(String[] args) {
   int a = 100;
   int b = 26;

   int max = Math.max(a, 4 * b);
   ...

}
```

```
public static int max(int x, int y) {
   if (x >= y) return x;
   return y;
}
```

*argument values*

100

104

*parameter variables*

*argument expressions*

104

*return value*

# Functions: quiz 1

**What does the following program print?**

**A.** -126

**B.** 126

**C.** Compile-time error.

**D.** Run-time error.

```
public class Mystery {

   public static void negate(int a) {
      a = -a;
   }

   public static void main(String[] args) {
      int a = 126;
      negate(a);
      StdOut.println(a);
   }
}
```

# Functions: quiz 2

**What does the following program print?**

**A.** 12 6

**B.** -12 -6

**C.** Compile-time error.

**D.** Run-time error.

```java
public class AnotherMystery {

   public static void negate(int[] b) {
      for (int i = 0; i < b.length; i++)
         b[i] = -b[i];
   }

   public static void main(String[] args) {
      int[] a = { 12, 6 };
      negate(a);
      StdOut.println(a[0] + " " + a[1]);
   }
}
```

# Side effects with arrays

## Functions and arrays.

- A function can have the side effect of changing the elements in an argument array.
- But the function cannot change the argument array itself.

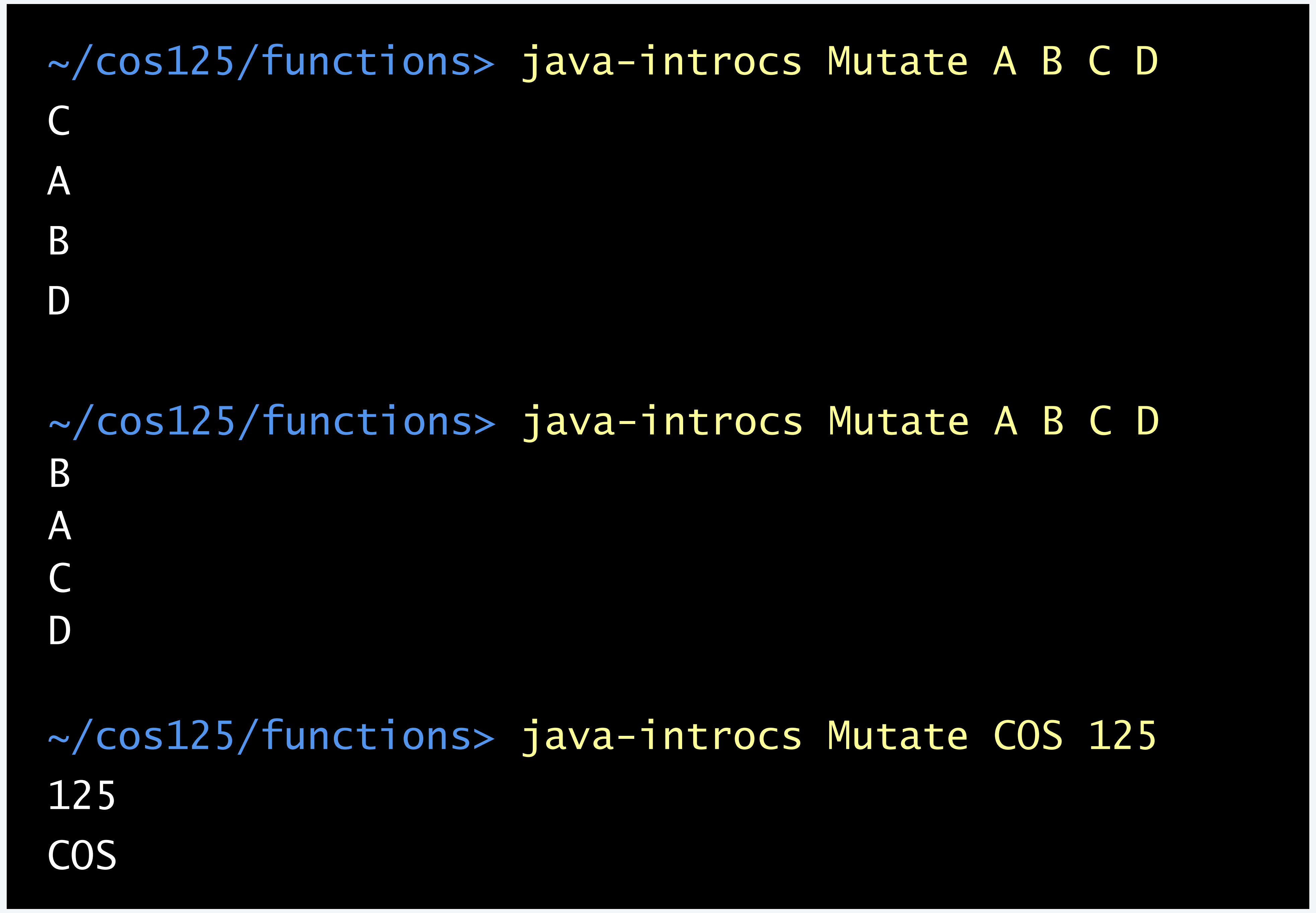
*shuffle, reverse, sort, shift, ...*

*to refer to a different array (e.g., of a different length or type)*

```
public class Mutate {

   public static void shuffle(String[] a) {
      int n = a.length;
      for (int i = 0; i < n; i++) {
         int r = (int) (Math.random() * (i + 1));
         String temp = a[r];
         a[r] = a[i];
         a[i] = temp;
      }
   }

   public static void main(String[] args) {
      shuffle(args);
      for (int i = 0; i < args.length; i++)
         StdOut.println(args[i]);
   }
}
```

*a[] and args[] refer to the same array*

*swaps a[r] and a[i]*

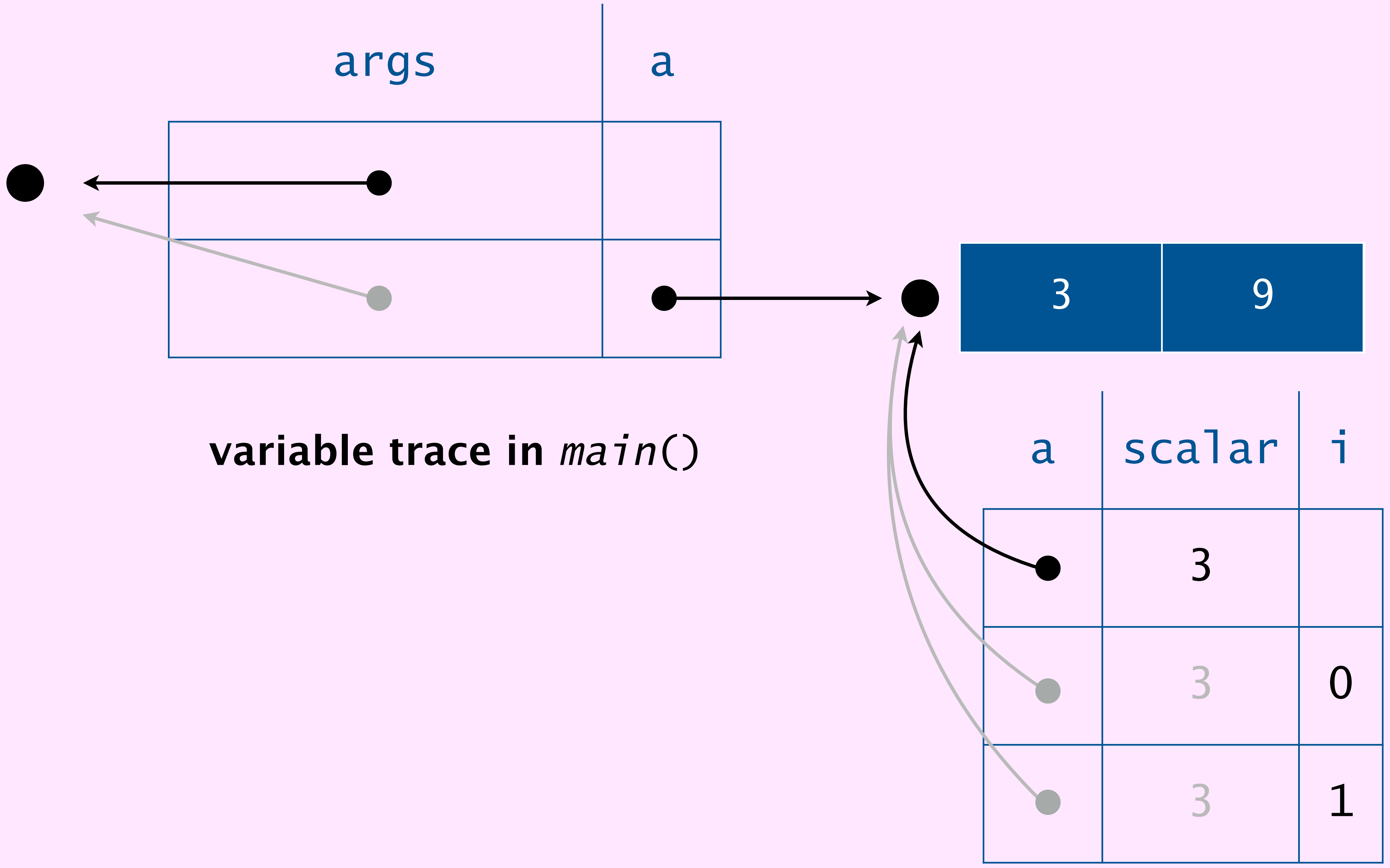


```
~/cos125/functions> java-introcs Mutate A B C D
C
A
B
D

~/cos125/functions> java-introcs Mutate A B C D
B
A
C
D

~/cos125/functions> java-introcs Mutate COS 125
125
COS
```

# Mechanics of function calls

```
public class Polynomial {
   public static void scalarMultiply(int[] a, int scalar) {
      for (int i = 0; i < a.length; i++)
         a[i] *= scalar;
   }

   public static void main(String[] args) {
      int[] a = { 1, 3 };
      scalarMultiply(a, 3);
      StdOut.println(a[0] + " " + a[1]);
   }
}
```

```
~/cos125/functions> java-introcs Polynomial
3 9
```



| args | a |
|---|---|
| ● | |
| ● | ● |

| 3 | 9 |
|---|---|

variable trace in *main*()

| a | scalar | i |
|---|---|---|
| ● | 3 | |
| ● | 3 | 0 |
| ● | 3 | 1 |

variable trace in *scalarMultiply*()

# Copying an array

Beware of common bugs!

```
public static int[] copy(int[] a) {
   return a;
}
```

✗

```
public static void copy(int[] a, int[] b) {
   b = a;
}
```

✗

```
public static int[] copy(int[] a) {
   int[] b = new int[a.length];
   for (int i = 0; i < a.length; i++)
      b[i] = a[i];
   return a;
}
```

✗

```
public static void copy(int[] a, int[] b) {
   for (int i = 0; i < a.length; i++)
      b[i] = a[i];
}
```

✓

↑

*if calling code ran b* `= new int[a.length]`

```
public static int[] copy(int[] a) {
   int[] b = new int[a.length];
   for (int i = 0; i < a.length; i++)
      b[i] = a[i];
   return b;
}
```

✓

```
public static void copy(int[] a, int[] b) {
   b = new int[a.length];
   for (int i = 0; i < a.length; i++)
      b[i] = a[i];
}
```

✗

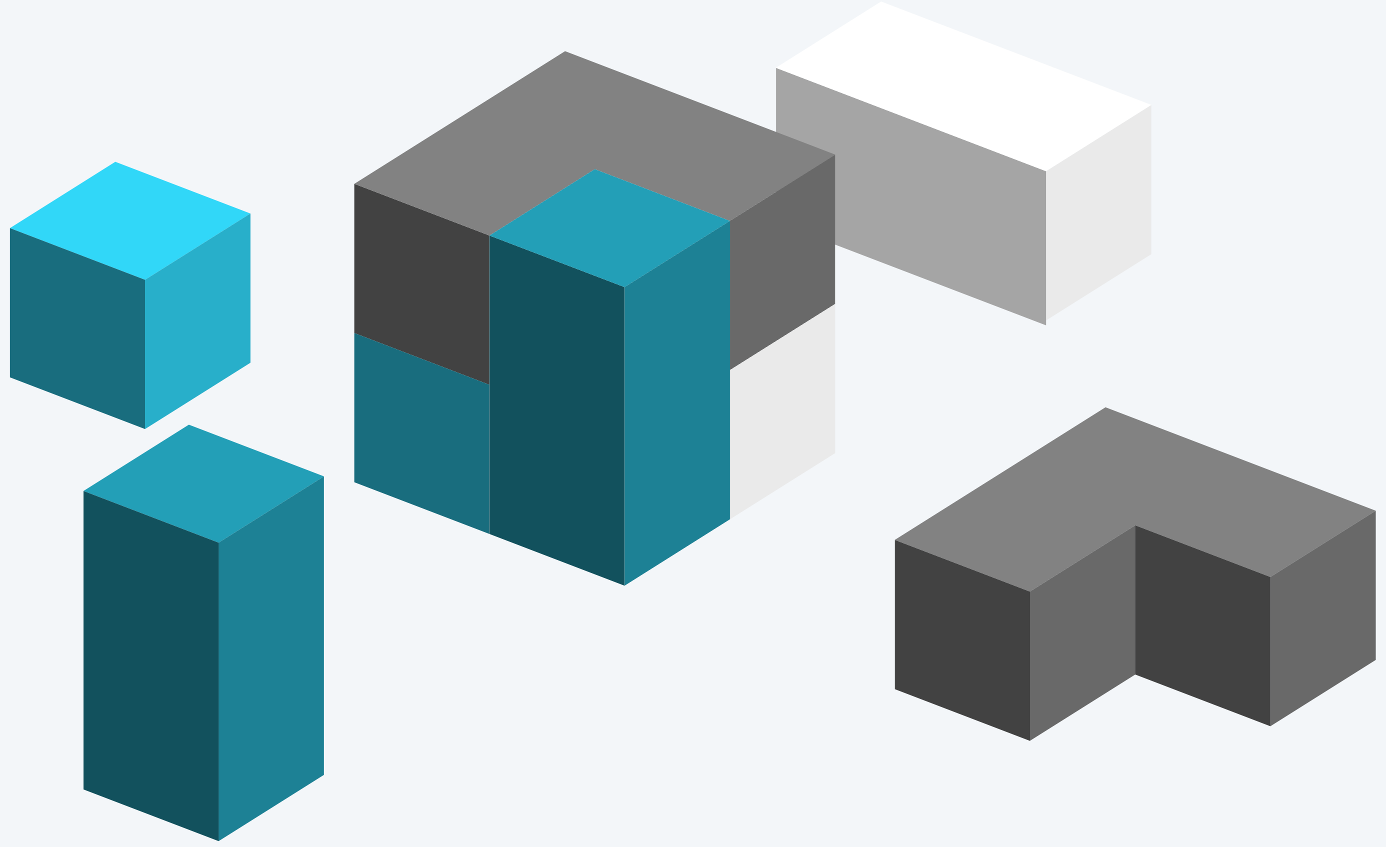
# Procedural decomposition

Decomposition. Break up a complex programming problem into smaller functional parts.

Procedural decomposition. Implement each part as a separate function.

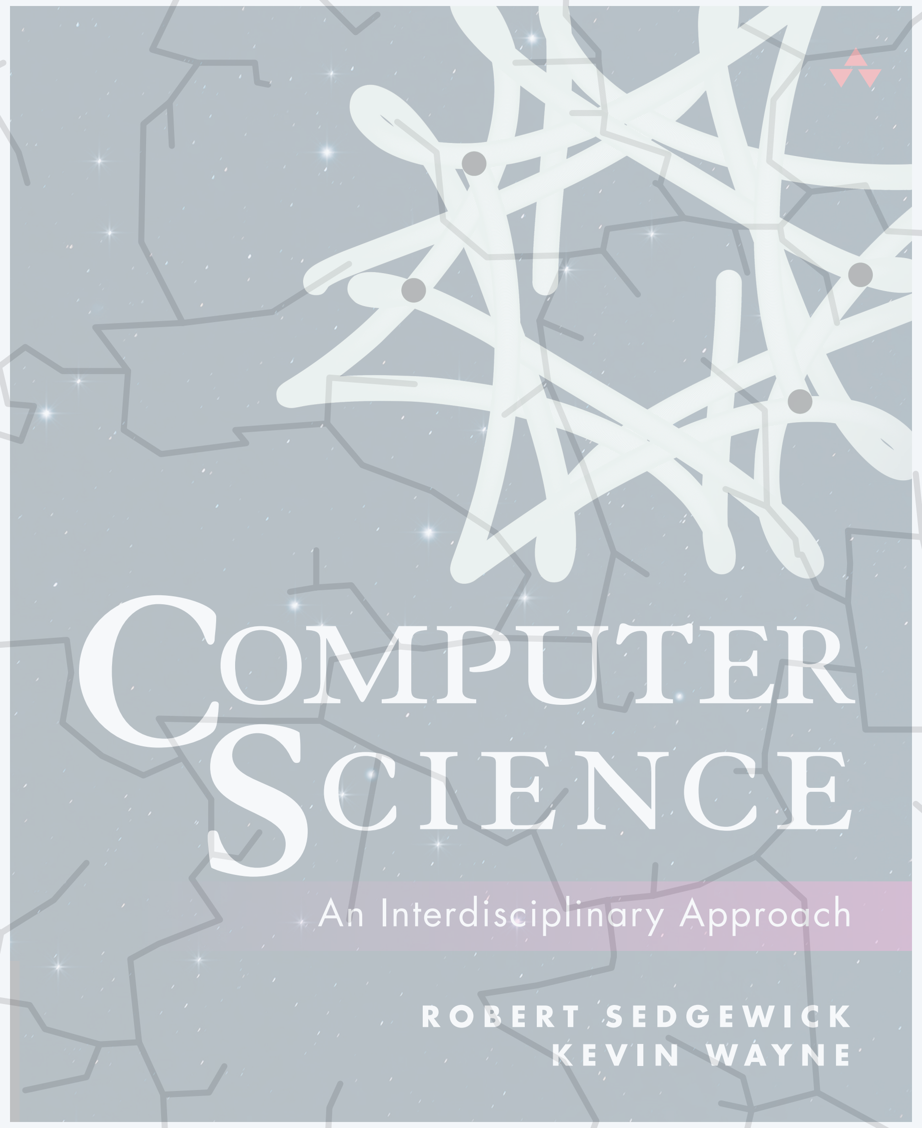
Example. Find the root of a polynomial.

- Approximate until convergence.
- Apply the Newton–Raphson iteration.
- Compute the derivative of a polynomial.
- Evaluate a polynomial at a point.



Benefits. Supports the 3 Rs:

- Readability: understand and reason about code.
- Reliability: test, debug, and maintain code.
- Reusability: reuse and share code.



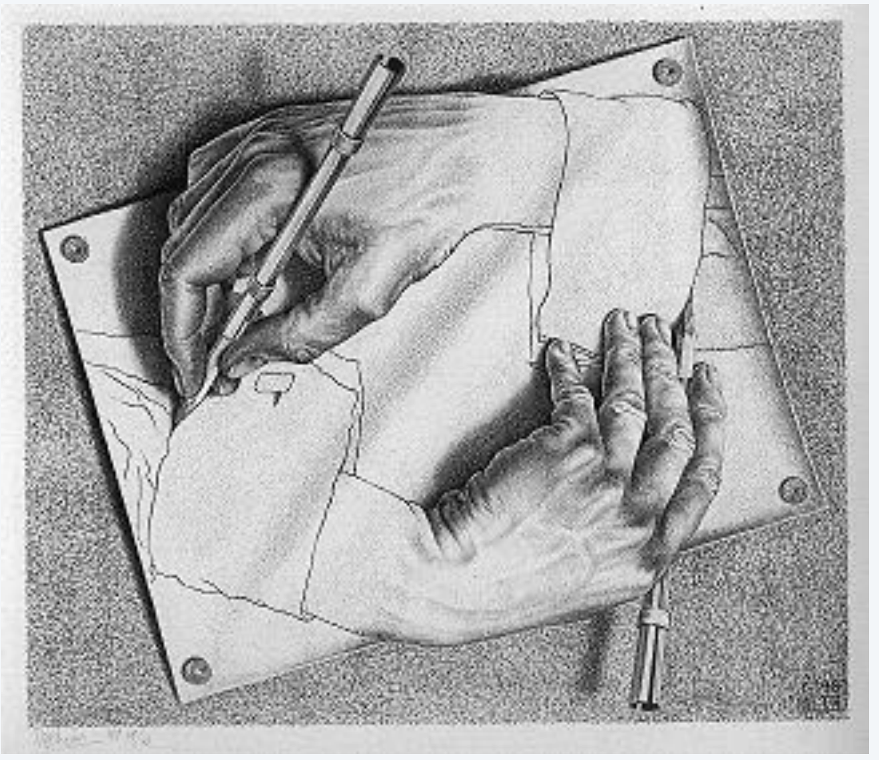
https://introcs.cs.princeton.edu

## 2.1 Functions

- ‣ *call by value*
- ‣ *recursion*
- ‣ *what next?*

# Overview

Recursion is when something is specified in terms of itself. ← *self-reference*



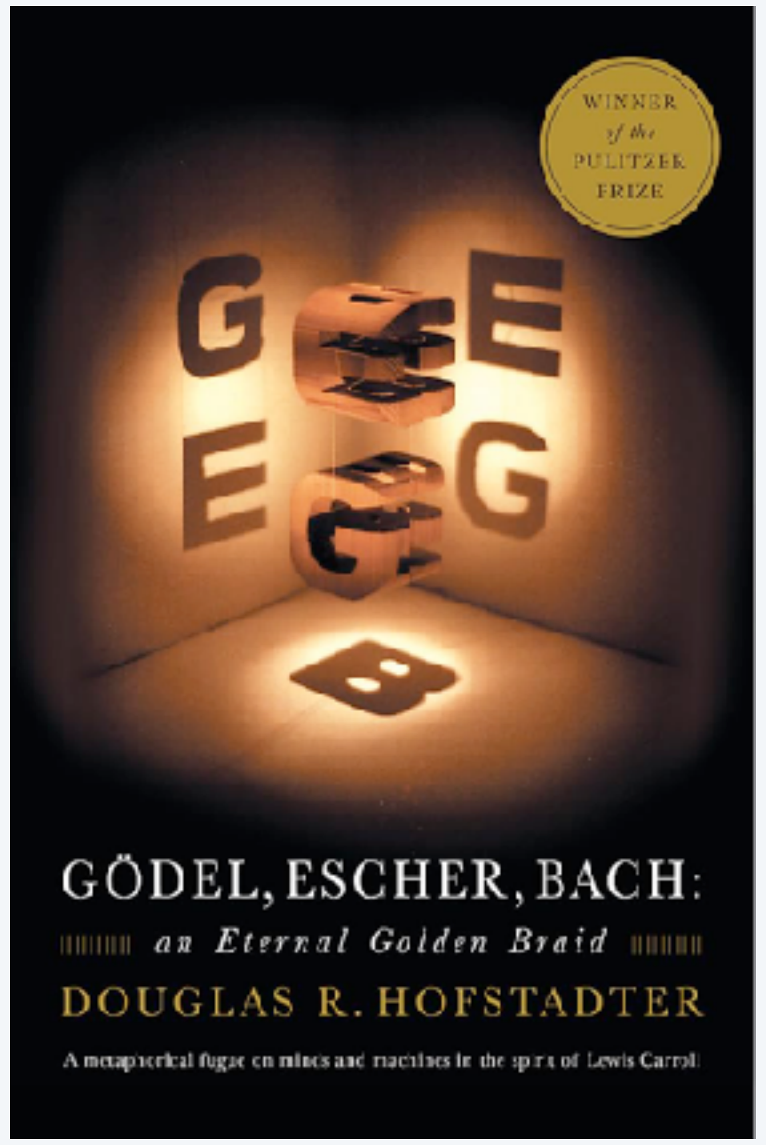
**Drawing Hands, by M. C. Escher**

## Why learn recursion?

- Powerful programming paradigm.
- Insight into the nature of computation and math. ← *proofs by induction, incompleteness theorems*

## Many computational artifacts are naturally self-referential.

- File system with folders containing folders.
- Binary trees.
- Divide-and-conquer algorithms.
- ⋮


WINNER of the PULITZER PRIZE
GÖDEL, ESCHER, BACH:
an Eternal Golden Braid
DOUGLAS R. HOFSTADTER
A metaphorical fugue on minds and machines in the spirit of Lewis Carroll

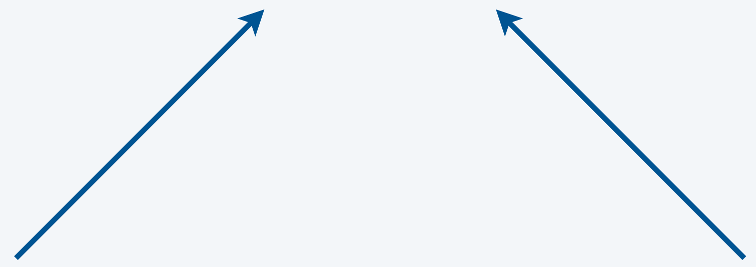
# Recursive functions

A recursive function calls itself.

- Base case: if the argument is "simple," compute directly.
- Reduction step: if the argument is "complicated," call function on simpler argument and "update."

Example: Factorial function $n! = n \cdot (n-1) \cdots 2 \cdot 1$.

- Base case: $0! = 1$ (by definition).
- Reduction step: $n! = (n-1)! \cdot n$.



*simpler argument*

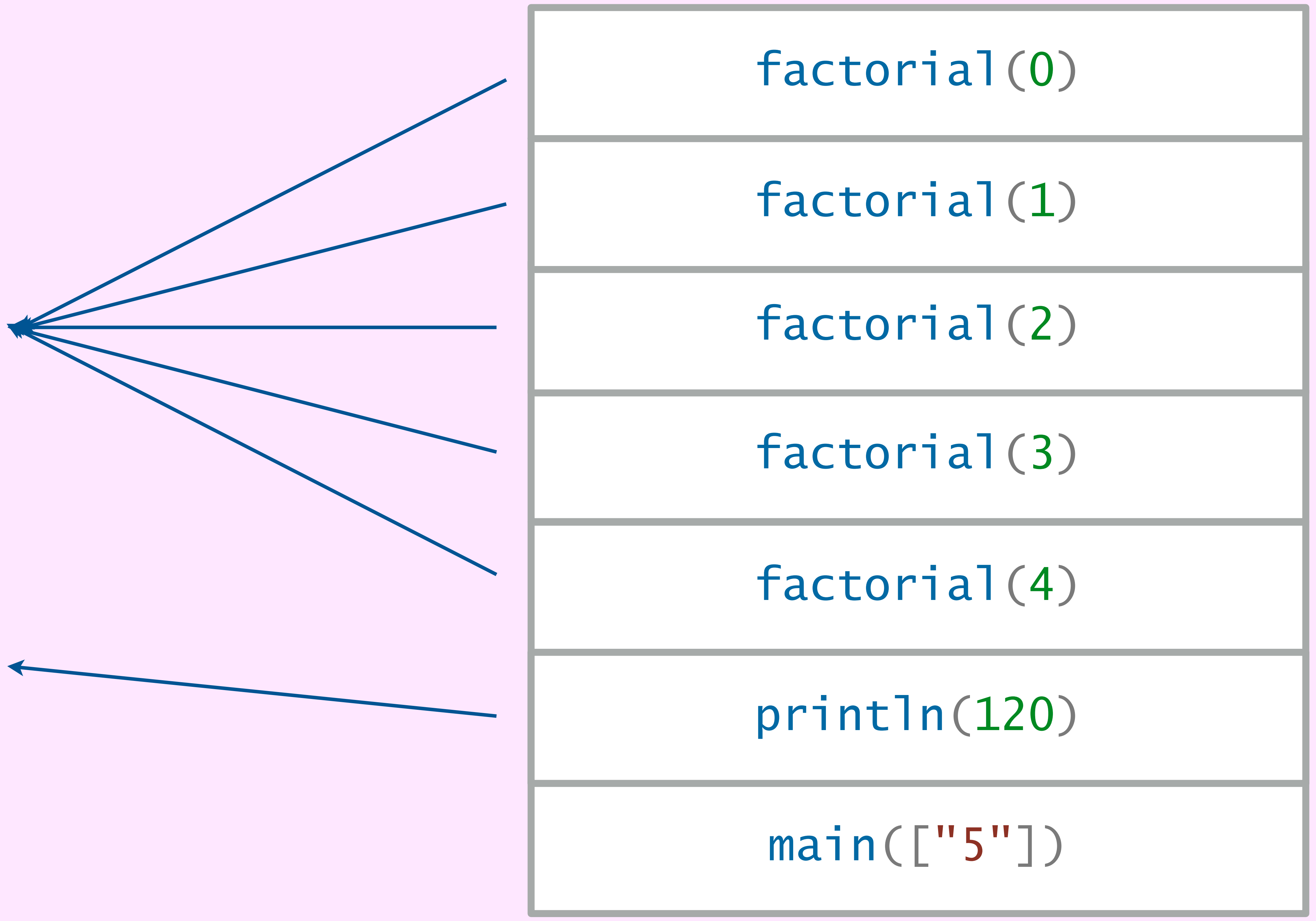
*update (multiply by n)*

```
public static int factorial(int n) {
   if (n == 0) return 1;
   else        return n * factorial(n - 1);
}
```

# Recursive function calls

```
public static int factorial(int n) {
   if (n == 0)
      return 1;
   else
      return n * factorial(n - 1);
}

public static void main(String[] args) {
   int n = Integer.parseInt(args[0]);
   StdOut.println(factorial(n));
}
```

```
~/cos125/functions> java-introcs Factorial 5
120
```



**function-call stack**

# Functions: quiz 3

**What does the following program print when $n = 4$?**

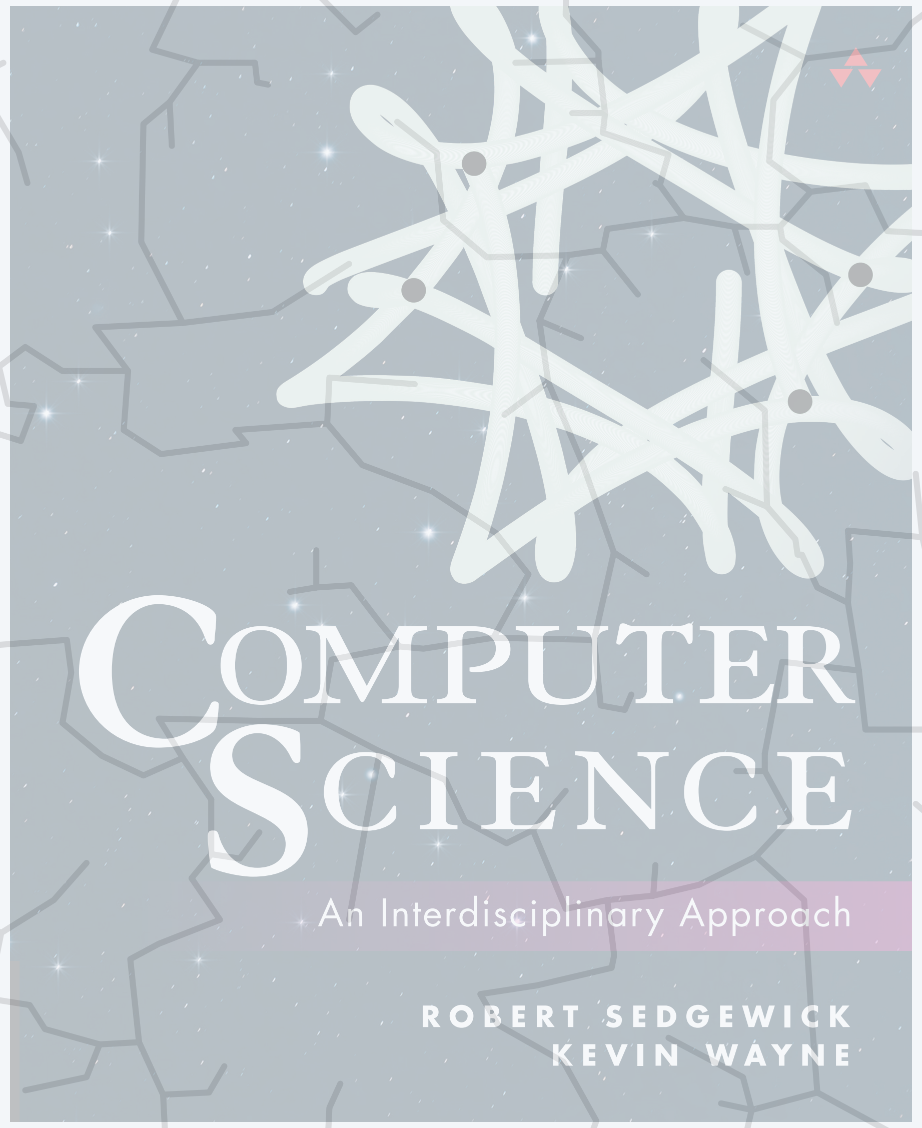
**A.** 120

**B.** 24

**C.** Compile-time error.

**D.** Run-time error.

```
public class YetAnotherMystery {
   public static int factorial(int n) {
      return n * factorial(n - 1);
   }

   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      StdOut.println(factorial(n));
   }
}
```



https://introcs.cs.princeton.edu

# 2.1 FUNCTIONS

- call by value
- recursion
- what next?

# Sorting algorithms

Goal. Place numbers of $n$-integer array in sorted order.

Solution. Mergesort: recursive with $n \log n$ runtime order of growth!

- Base case: if array has length 1, return it.
- Reduction step: divide array in half; sort both halves then merge.

```
public static void sort(int[] a, int lo, int hi) {
   if (hi <= lo) return;
   int mid = (lo + hi) / 2;
   sort(a, lo, mid);
   sort(a, mid + 1, hi);
   merge(a, lo, mid, hi);
}
```

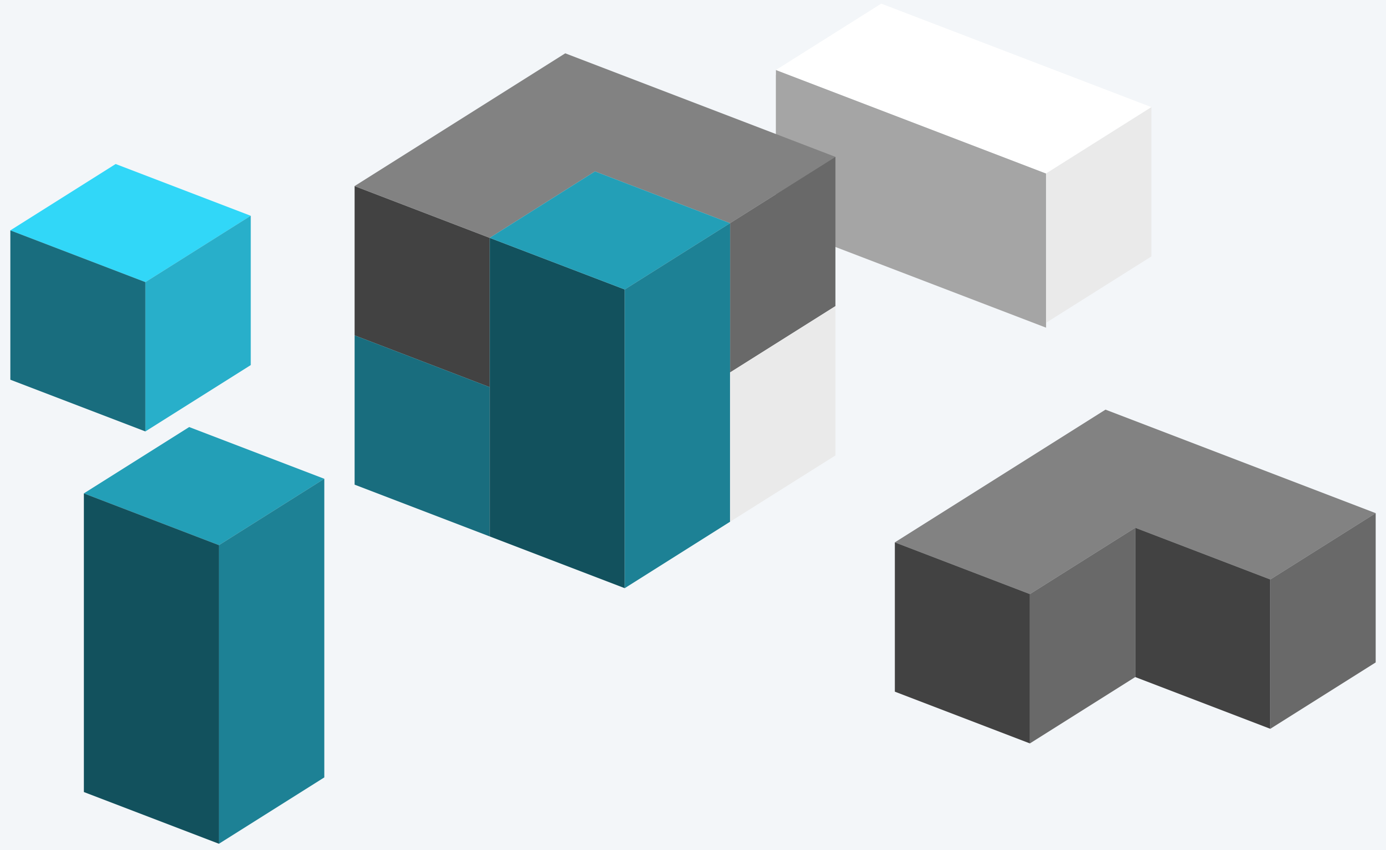
# Object-oriented programming

Data type. A set of values and a set of operations on those values.

Java class. Java’s mechanism for defining a new data type.

Object. An instance of a data type that has

- State: value from its data type.
- Behavior: actions defined by the data type’s operations.
- Identity: unique identifier (e.g. memory address).



```java
public class PrintPoly{
   public static void main(String[] args) {
      Polynomial p = new Polynomial(1.0, 1.0);
      double[] c = new double[] {1.0, -1.0};
      Polynomial q = new Polynomial(c);
      p.multiplyBy(q);
      p.print();
   }
}
```

```
~/cos125/functions> java-introcs PrintPoly
1.0 * X^2 - 1.0
```

# Theory of computing

Scenario 1. You just wrote a program that solves Problem A. You're feeling proud (as you should), and think your program is the best.

Can you prove it's the best solution for Problem A?

Scenario 2. You spent hours and hours trying to solve Problem B, but didn't get there. You're smart and know it — so Problem B looks like the issue.

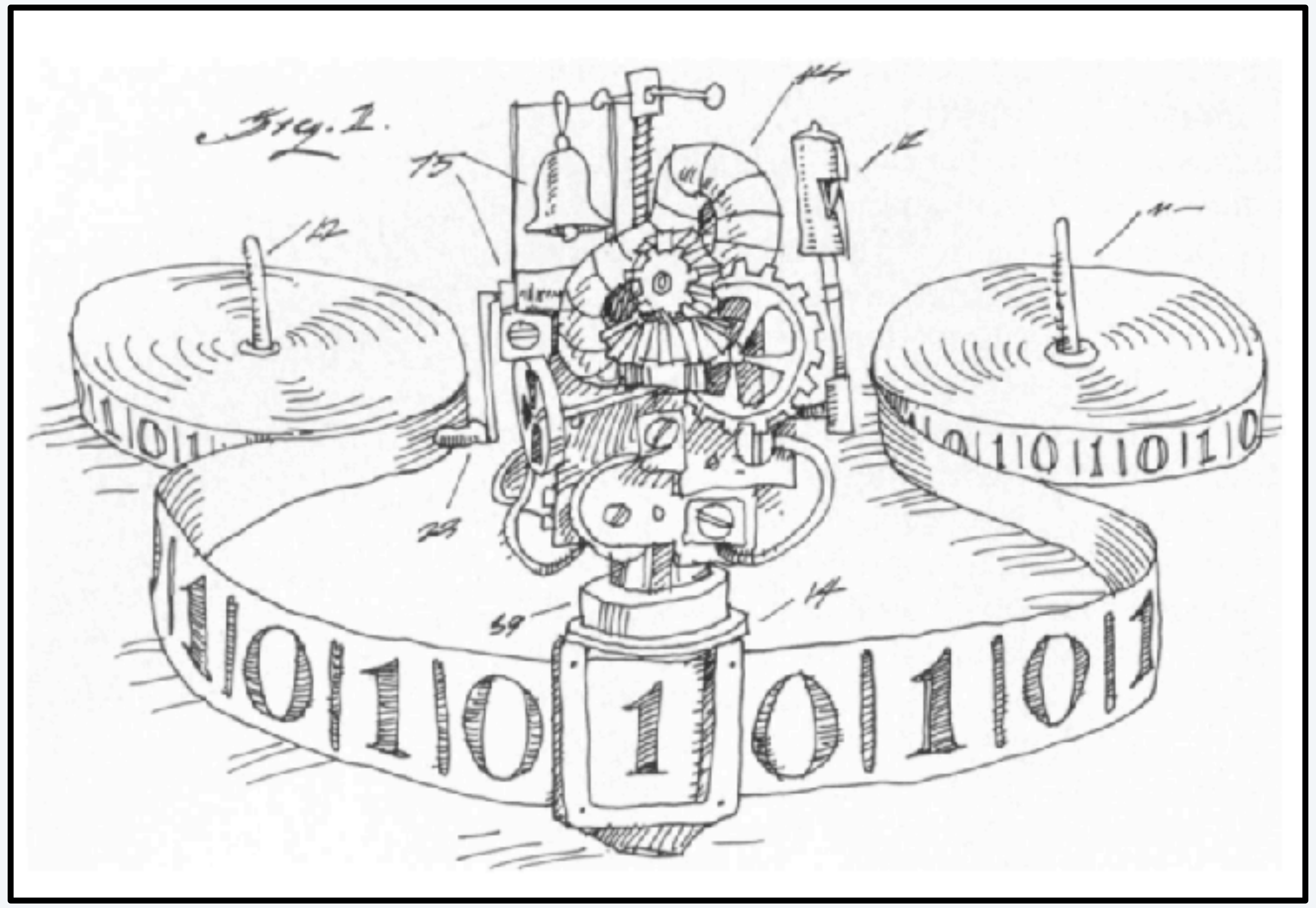
Can you prove Problem B is really hard to solve?

# Fundamental questions

Q1. What is an algorithm?

Q2. What is an efficient algorithm?

Q3. Which problems can be solved efficiently?



**A Turing machine**

# Final exam

Day: August 14th

Place: McDonnell 105

Time: 1:30pm to 2:50pm

8 quiz-type questions (so 10min/question, on average).

Closed book, but can bring "cheatsheet:"

- 8.5-by-11 paper, one side, in your own handwriting.

**Good luck!**

Study material:

- Review quiz
- Textbook
- Ed

# Credits

| media | source | license |
|---|---|---|
| *Gears* | Adobe Stock | education license |
| *Function Gradient* | Adobe Stock | education license |
| *Function Machine* | Wvbailey | public domain |
| *Gödel, Escher, Bach cover* | Amazon | |
| *Drawing Hands* | Wikipedia | |
| *Happy Programmer* | Adobe Stock | education license |
| *Frustrated Programmer* | Adobe Stock | |
| *Cartoon of Turing Machine* | Tom Dunne | |