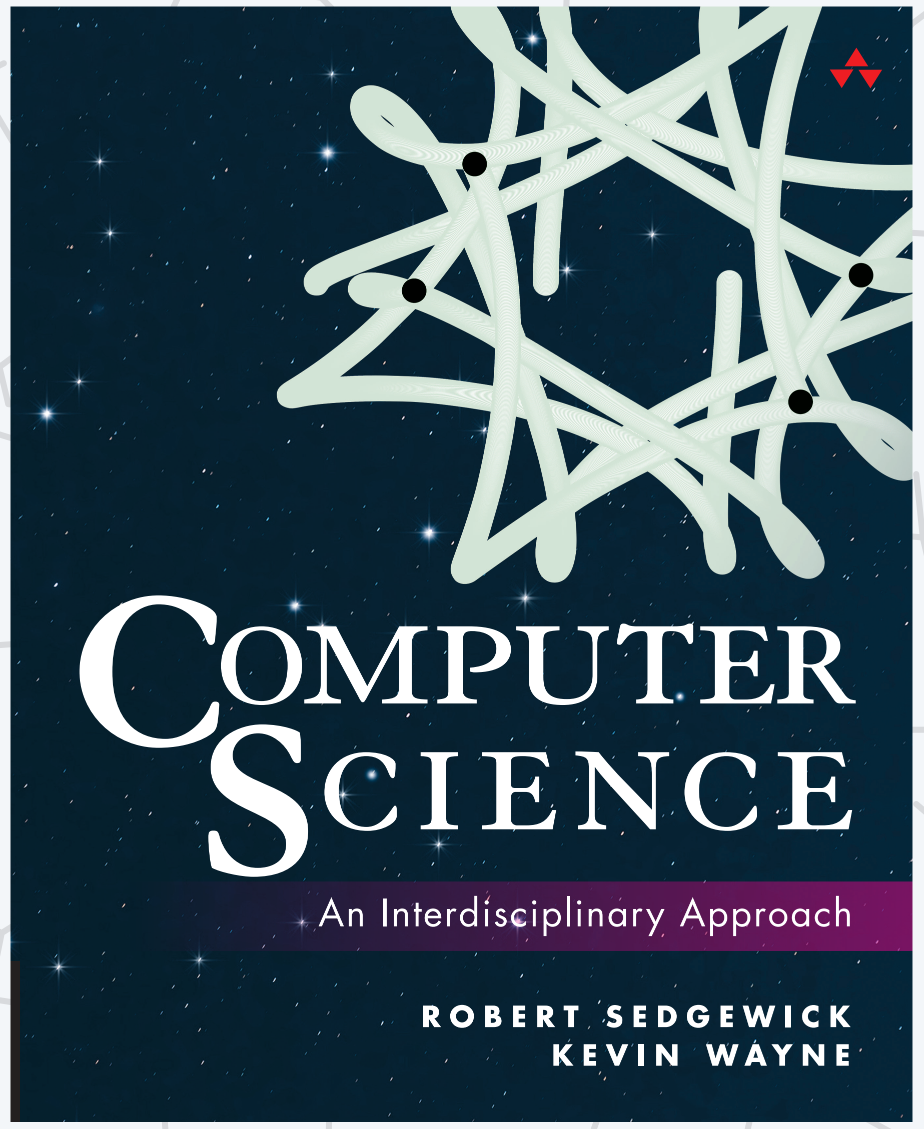


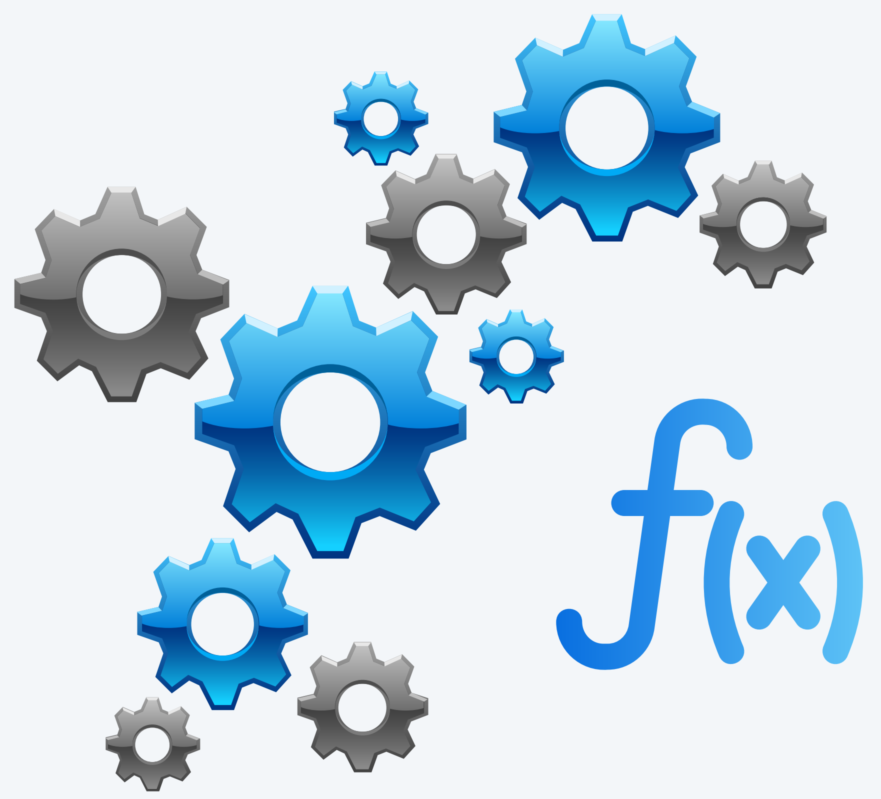
https://introcs.cs.princeton.edu

2.1 FUNCTIONS

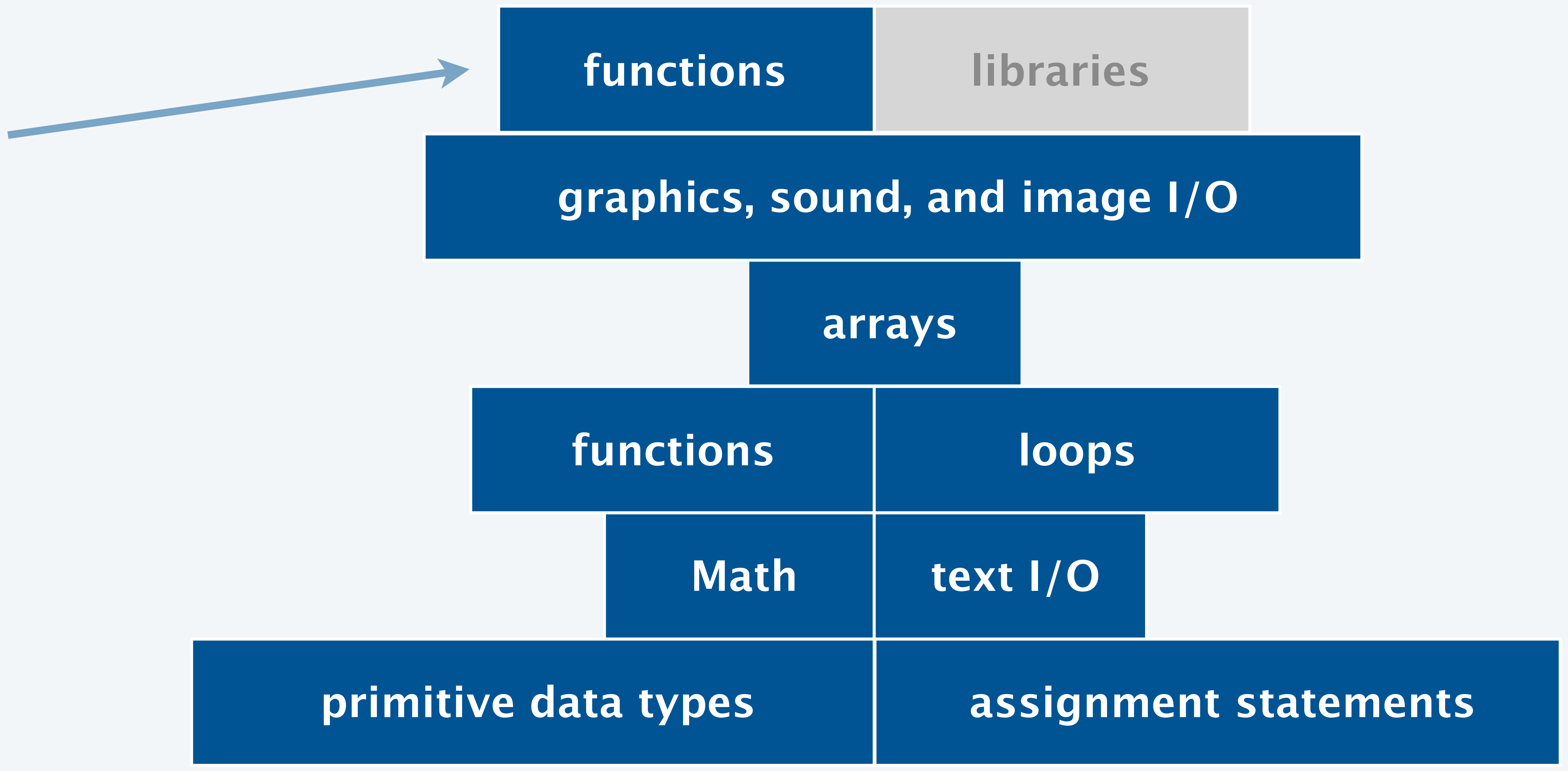
- *flow-of-control*
- *properties*
- *call stack and scope*
- *APIs and libraries*

Last updated on 8/6/24 11:47AM

Basic building blocks for programming



divide a program into functions

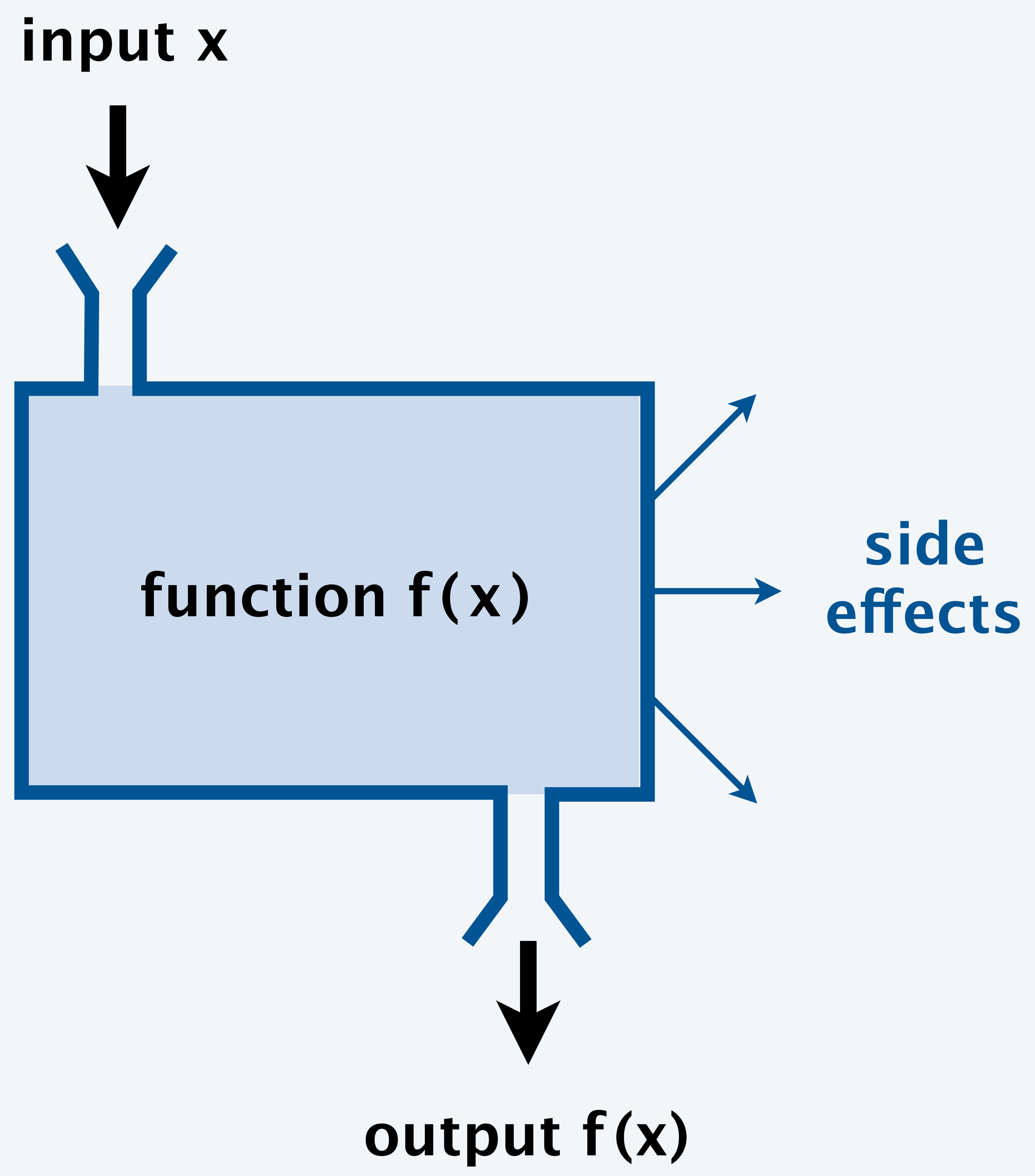


Functions

Java function (static method).

- Takes zero or more input arguments.
- Returns zero or one output value.
- May cause side effects.

more general than mathematical functions



Benefits. Makes code easier to read, test, debug, reuse, and extend.

Familiar examples.

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

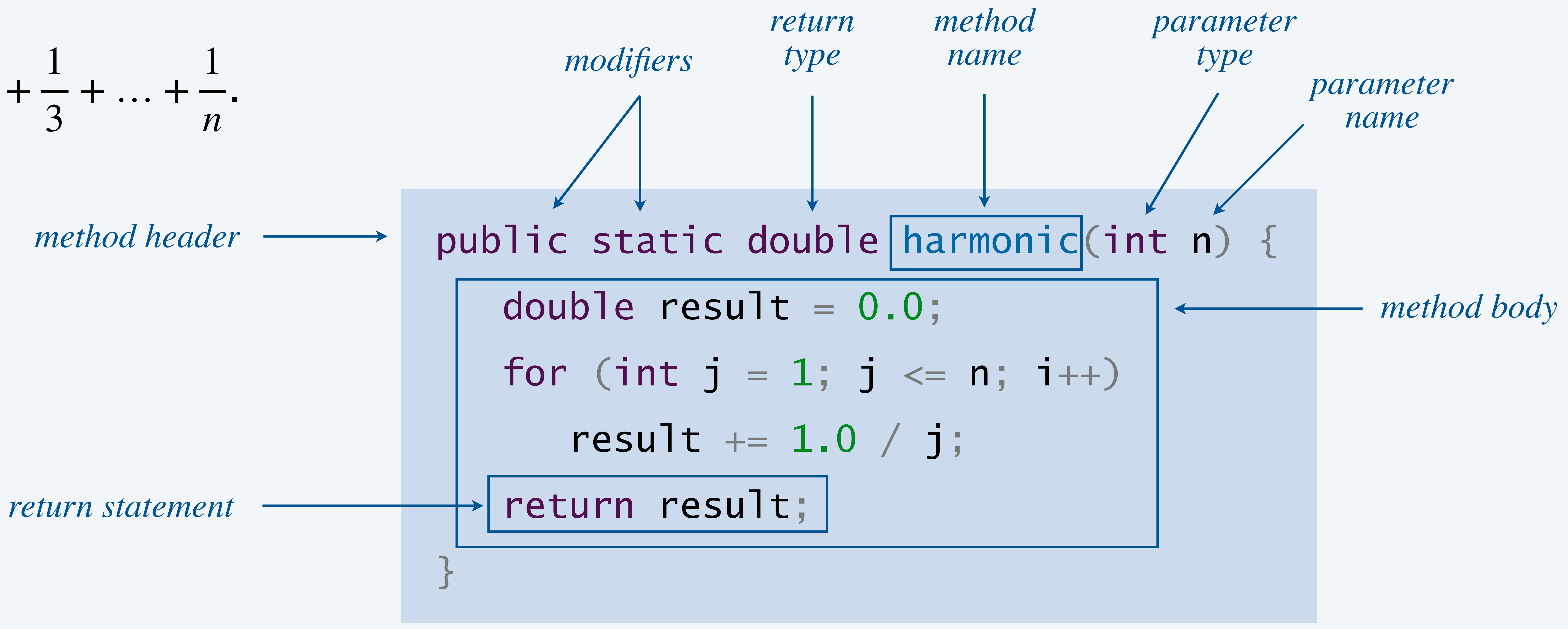
Anatomy of a Java function (static method)

To implement a Java function:

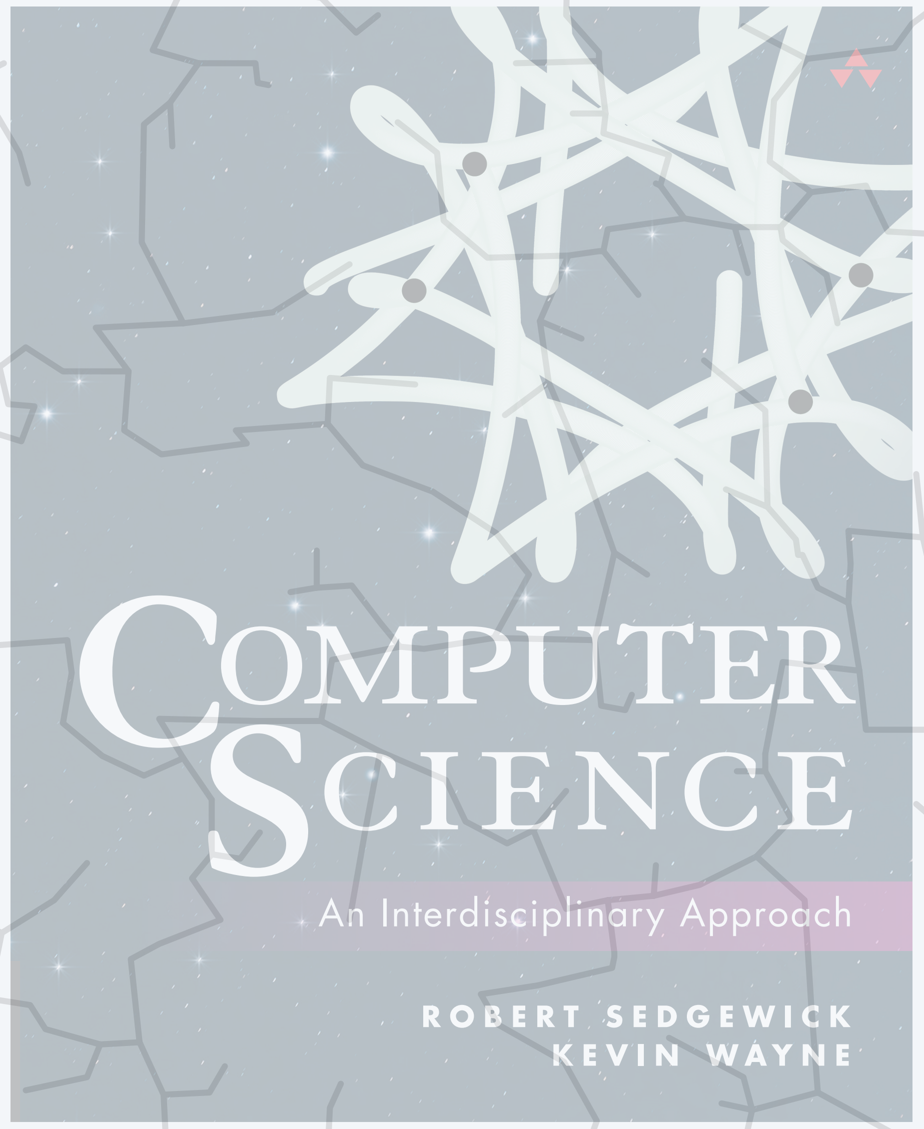
- Choose a *method name*.
- Declare type and name of each *parameter variable*.
- Specify type for *return value*.
- Include *modifiers*. ← *for now, always* `public` *and* `static`
- Implement *method body*,
  including a *return statement*.

← *method header*

Ex. Harmonic sum: $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$.



```
public static double harmonic(int n) {
   double result = 0.0;
   for (int j = 1; j <= n; i++)
      result += 1.0 / j;
   return result;
}
```



https://introcs.cs.princeton.edu

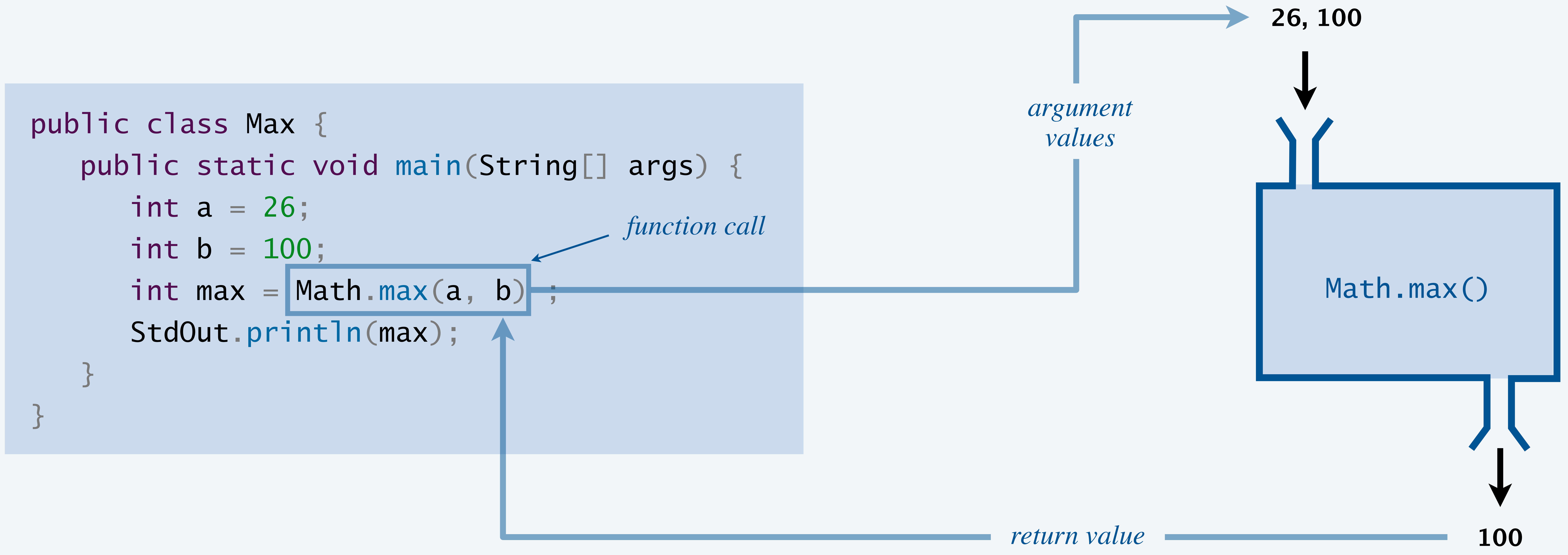
2.1 Functions

- *flow-of-control*
- *properties*
- *call stack and scope*
- *APIs and libraries*

Flow of control

Mechanics of a function call.

- Control transfers from calling code to function code, passing argument values.
- Function code executes, producing a return value.
- Control transfers back to calling code. ← *function-call expression evaluates to return value*



```
public class Max {
   public static void main(String[] args) {
      int a = 26;
      int b = 100;
      int max = Math.max(a, b);
      StdOut.println(max);
   }
}
```

Bottom line. Functions provide a useful *way* to control the flow of execution.

Function call trace

```
public class MaxMany {
   public static int max(int a, int b) {
      if (a > b)
         return a;
      else
         return b;
   }

   public static void main(String[] args) {
     int result = Integer.parseInt(args[0]);
     for (int i = 1; i < args.length; i++)
         result = max(result, Integer.parseInt(args[i]));
     StdOut.println(result);
   }
}
```

args	i	result
["1", "5", "3"]		
["1", "5", "3"]		1
["1", "5", "3"]	1	1
["1", "5", "3"]	1	5
["1", "5", "3"]	2	5
["1", "5", "3"]	2	5
["1", "5", "3"]		5

variable trace in *main*()

```
~/cos125/functions> java-introcs MaxMany 1 5 3
5
```

Functions: quiz 1

What is the result of executing this program with the given command-line argument?

A. 10

B. 11

C. Compile-time error.

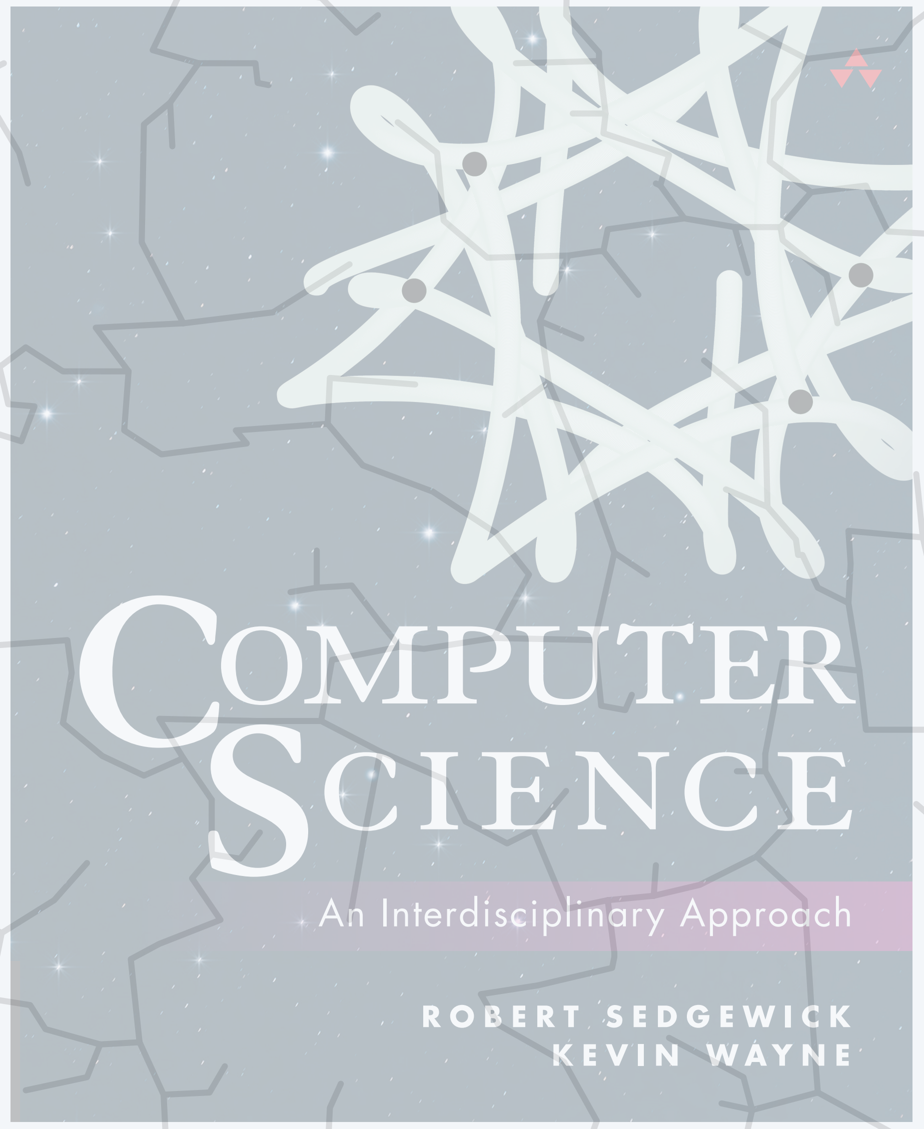
D. Run-time error.

```java
public class Mystery {

   public static int increment(int x) {
      return x + 1;
   }

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

```
~/cos125/functions> java-introcs Mystery 10
```


https://introcs.cs.princeton.edu

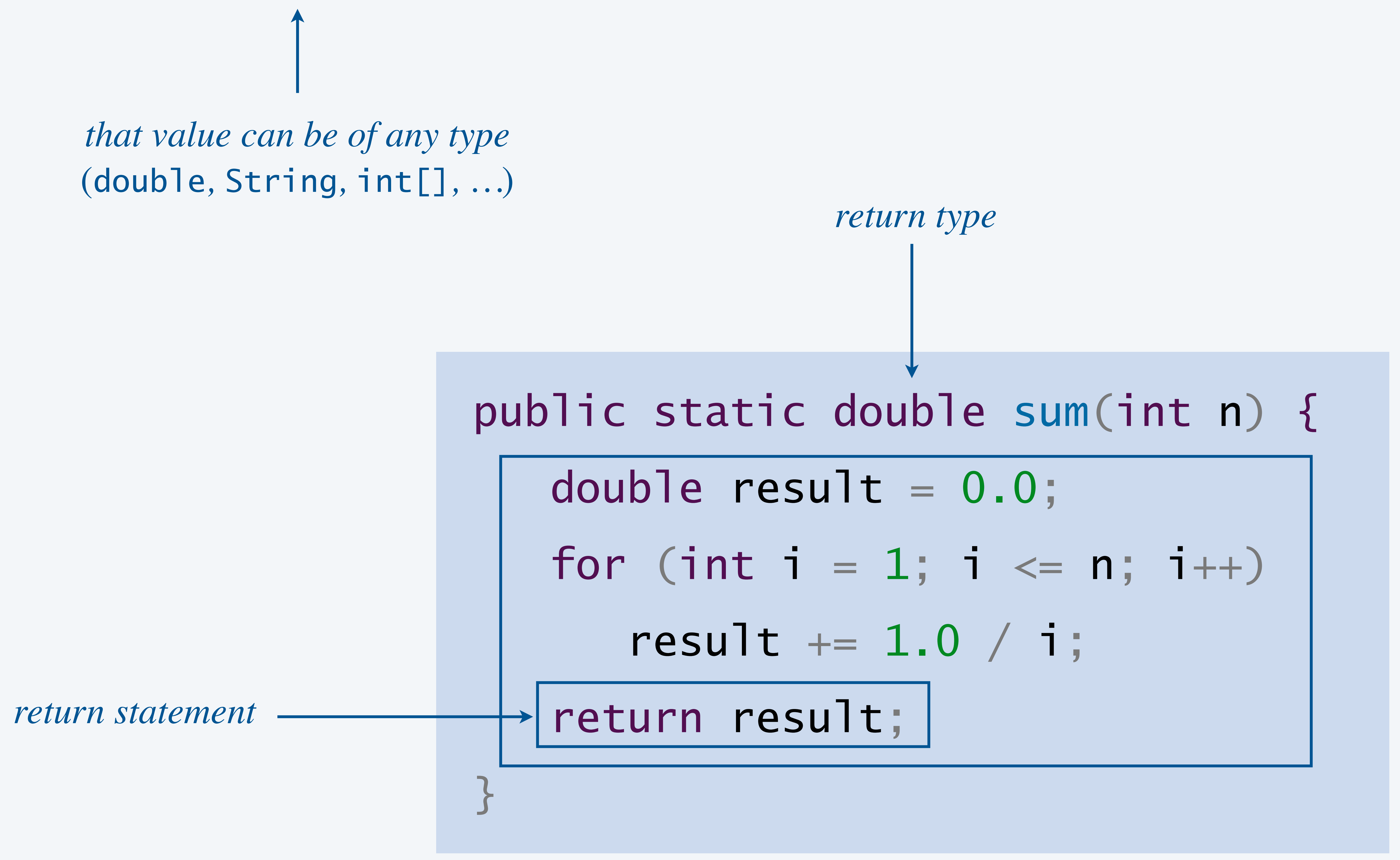
2.1 Functions

- ‣ *flow-of-control*
- ‣ *properties*
- ‣ *call stack and scope*
- ‣ *APIs and libraries*

Single *return* value

When a function reaches a return statement, it transfer control back to code that invoked it.

- The type of the return value must be compatible with the function's return type.
- Java returns a single return value to the calling code.



```
public static double sum(int n) {
   double result = 0.0;
   for (int i = 1; i <= n; i++)
      result += 1.0 / i;
   return result;
}
```

Multiple *return* statements

Control is transferred back to calling code upon reaching first *return* statement.

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

absolute value function

multiple `return` *statements*

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

equivalent function

Multiple arguments

A function can take multiple arguments.

- Each parameter variable has a type and a name.
- The argument values are assigned to the corresponding parameter variables.

Ex. Polynomial evaluation: $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$.

```
eval([1.0, 2.0, 1.0], 1.0)
```

```
public static double eval(double[] a, double x) {
   double result = 0.0, monomial = 1.0;
      for (int i = a.length - 1; i >= 0; i--, monomial *= x)
         result += a[i] * monomial;
   return result;
}
```

function takes one `double[]` *and one* `double` *argument*

Void functions

A method need not return a value.

- Its purpose is to produce side effects.
- Use keyword *void* as return type.
- No explicit *return* statement needed. ← *upon reaching the end of method, control returns to calling code*

```
public static void loop(String filename, int n) {
   for (int i = 0; i < n; i++) {
      StdAudio.play(filename);
   }
}
```

loop an audio file n times

```
public static void main(String[] args) {
   int n = Integer.parseInt(args[0]);
   if (n <= 0) {
      StdOut.println("n must be positive");
      return;
   }
   ...
}
```

abort if the wrong number of command-line arguments

Multiple functions

You can define many functions in a class.

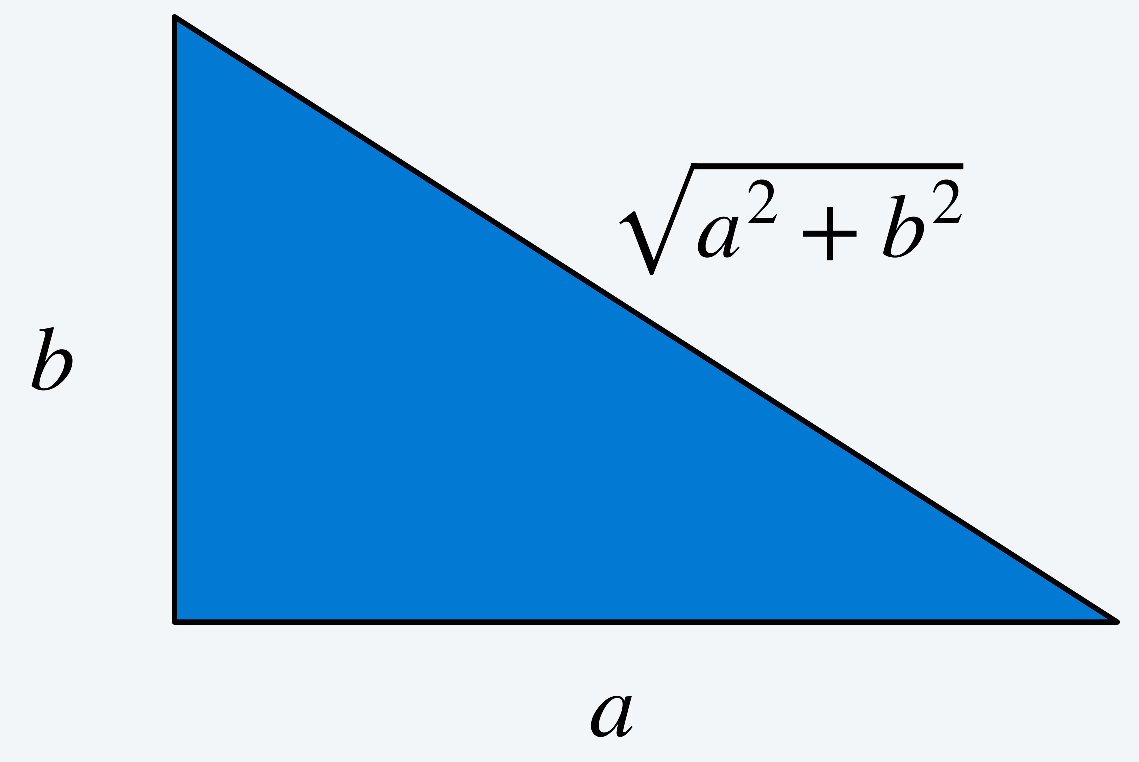
- One function can call another function.
- The order in which the functions are defined in the file is unimportant.

```
public class RightTriangle {
   public static double square(double x) {
      return x*x;
   }

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

function calls a function defined in a different class

function calls a function defined in the same class



Overloaded functions

Overloading. Two functions with the same name (but different ordered list of parameter types).

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

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

`abs(-126)` *calls this function (and evaluates to* `126`*)*

`abs(-126.0)` *calls this function (and evaluates to* `126.0`*)*

Note. These two overloaded functions appear in Java's *`Math`* library.

Another example: *`StdAudio.play(String filename)`* and *`StdAudio.play(double[] samples)`*

Overloaded functions

Overloading. Two functions with the same name (but different ordered list of parameter types).

```
public class Polynomial {
   public static double eval(double[] a, double x) {
      double result = 0.0, monomial = 1.0;
         for (int i = a.length - 1; i >= 0; i--, monomial *= x)
            result += a[i] * monomial;
      return result;
   }

   public static int eval(int[] a, int x) {
      int result = 0, monomial = 1;
         for (int i = a.length - 1; i >= 0; i--, monomial *= x)
            result += a[i] * monomial;
      return result;
   }
}
```

`evaluate(new double[] {1.0, -2.0, 1.0}, 1.0)`
calls this function (and evaluates to `0.0`*)*

`evaluate(new int[] {1, -2, 1}, 1)`
calls this function (and evaluates to `0`*)*

Functions: quiz 2

Which value does *`eval(new double[] {1.0, 0.0, 0.0}, 2)`* **return?**

A. `2.0`

B. `4.0`

C. `4`

D. Compile-time error.

E. Run-time error.

Functions: quiz 3

Which value does *`eval(new int[] {1, 0, 0}, 2.0)`* **return?**

A. `2.0`

B. `4.0`

C. `4`

D. Compile-time error.

E. Run-time error.

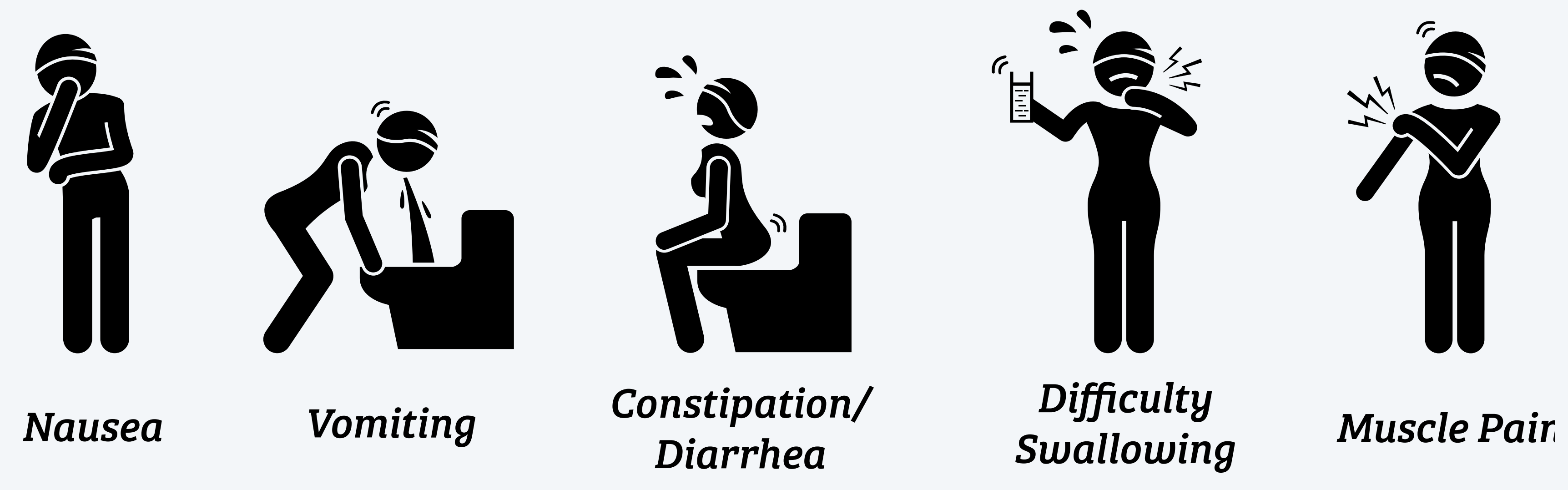
Side effects

Def. A side effect of a method is anything it does besides computing and returning a value.

- Print to standard output.
- Draw a circle.
- Play an audio file.
- Display a picture.
- Launch a missile.
- Consume input.
- Mutate an array.
- …

← *produce output*

← *stay tuned*



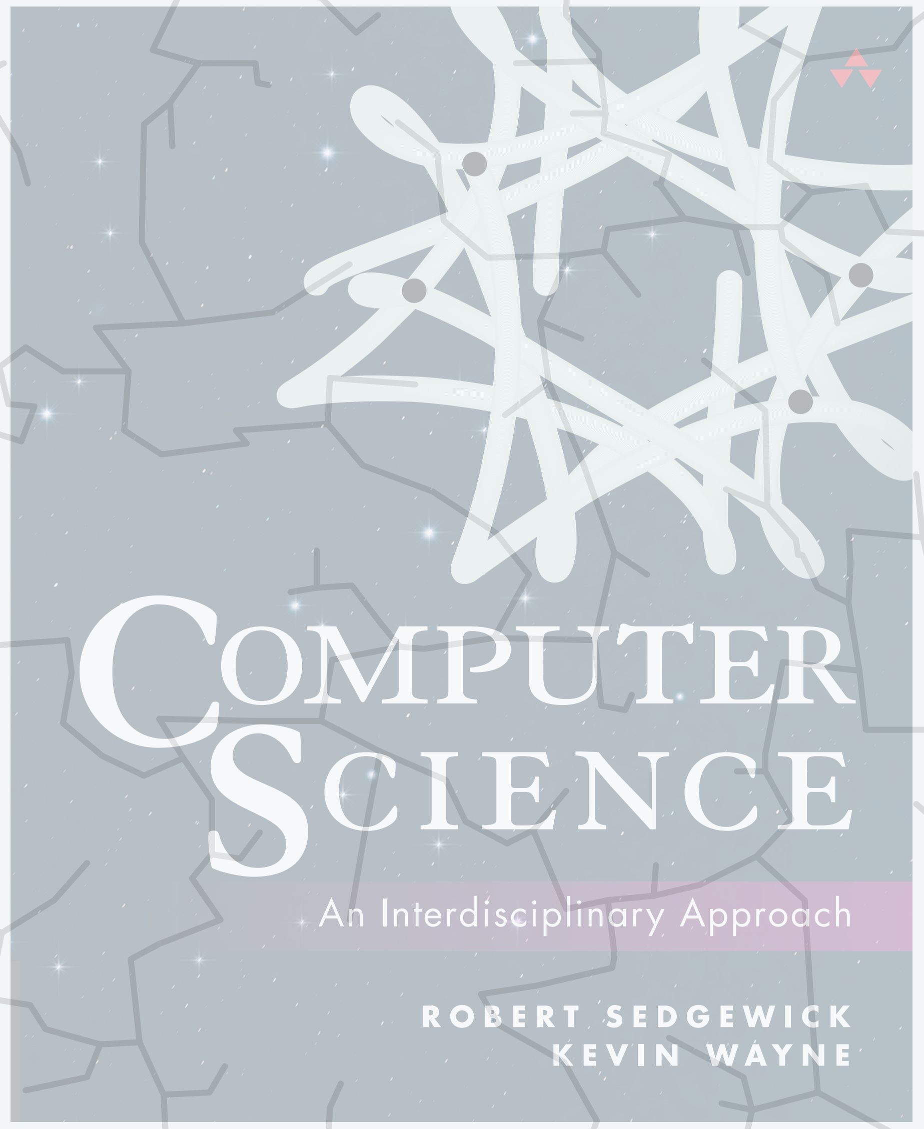
Note. The primary purpose of some methods is to produce side effects, not return values.

differs from medicine

Functions: quiz 4

Which of these functions both produces a side effect and returns a value?

- **A.** *`Integer.parseInt()`*
- **B.** *`StdAudio.play()`*
- **C.** *`StdIn.readInt()`*
- **D.** All of the above.
- **E.** None of the above.



https://introcs.cs.princeton.edu

2.1 Functions

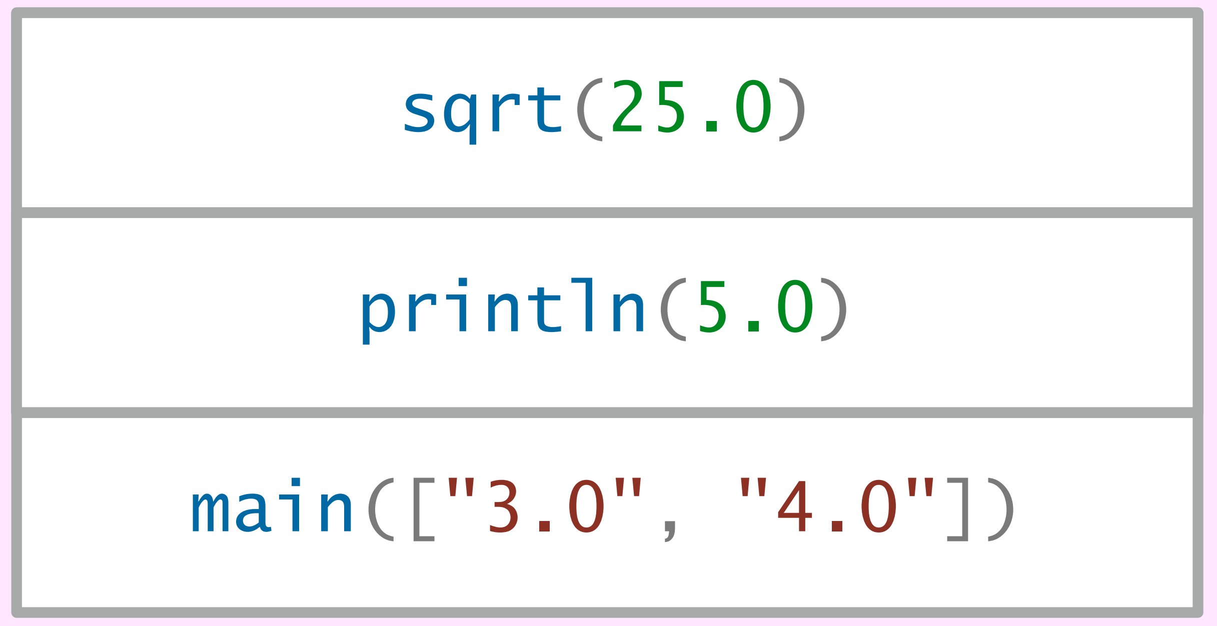
- ‣ *flow-of-control*
- ‣ *properties*
- ‣ *call stack and scope*
- ‣ *APIs and libraries*

Mechanics of function calls

```java
public class RightTriangle {
 public static double square(double x) {
   return x*x;
  }

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

  public static void main(String[] args) {
   int a = Double.parseDouble(args[0]);
   int b = Double.parseDouble(args[1]);
   StdOut.println(hypotenuse(a, b));
  }
}
```



function-call stack

Function-call trace

Function-call trace.

- Print name and argument values when each function is called.
- Print function's return value just before returning.
- Add indentation on function calls and subtract on returns.

```
public class RightTriangle {
  public static double square(double x) {
      return x*x;
   }

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

   public static void main(String[] args) {
      int a = Double.parseDouble(args[0]);
      int b = Double.parseDouble(args[1]);
      StdOut.println(hypotenuse(a, b));
   }
}
```

```
main("3.0", "4.0")
   parseDouble("3.0")
      return 3.0
   parseDouble("4.0")
      return 4.0
   hypotenuse(3.0, 4.0)
      square(3.0)
         return 9.0
      square(4.0)
         return 16.0
      sqrt(25.0)
         return 5.0
      return 5.0
   println(5.0)
      return
   return
```

function-call trace for ***RightTriangle***

Functions: quiz 5

Which value does *cube*(3) return?

A. 0.0

B. 1.0

C. 27.0

D. Compile-time error.

E. Run-time error.

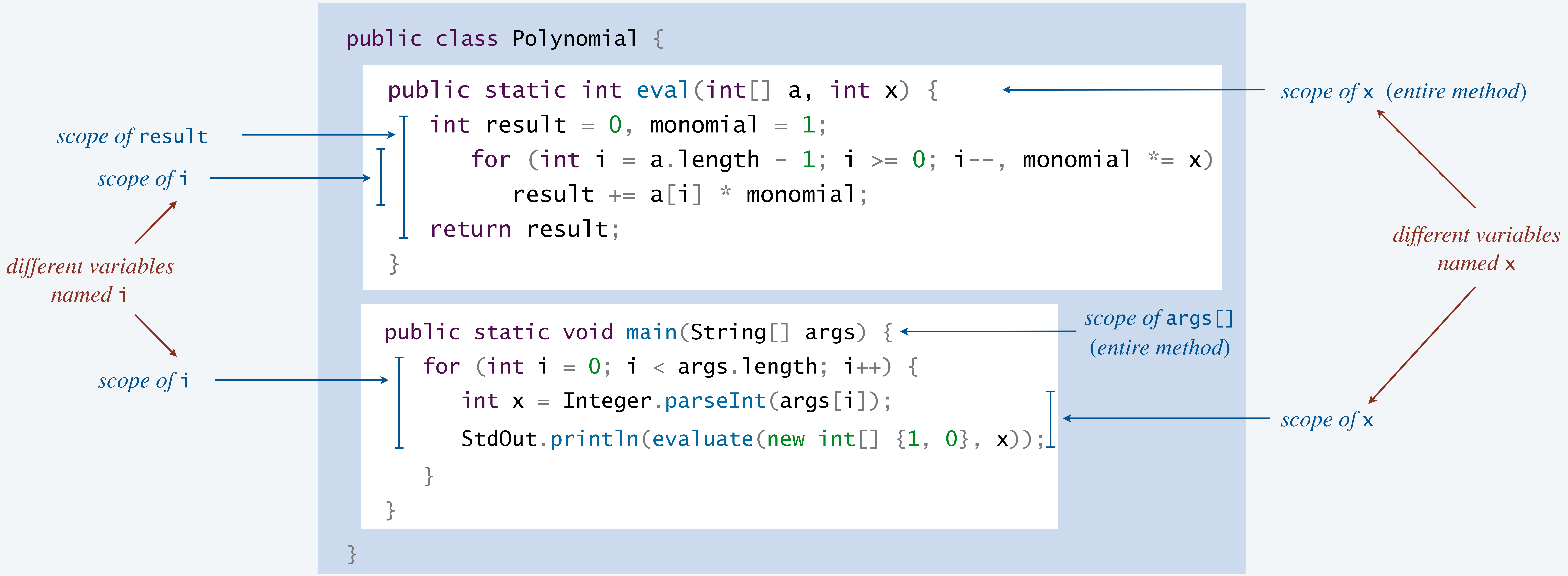
```
public static double cube(double i) {
   i = i * i * i;
   return i;
}
```

Scope of a variable

Def. The scope of a variable is the code that can refer to it by name. ← *code following its declaration, in the same block*

Significance. Can develop functions independently. ← *variables defined in one function do not interfere with variables defined in another*

Best practice. Declare variables so as to limit their scope.



```
public class Polynomial {
  public static int eval(int[] a, int x) {
     int result = 0, monomial = 1;
        for (int i = a.length - 1; i >= 0; i--, monomial *= x)
           result += a[i] * monomial;
     return result;
  }

  public static void main(String[] args) {
     for (int i = 0; i < args.length; i++) {
        int x = Integer.parseInt(args[i]);
        StdOut.println(evaluate(new int[] {1, 0}, x));
     }
  }
}
```

scope of x (entire method)
scope of result
scope of i
different variables named i
scope of i
different variables named x
scope of args[] (entire method)
scope of x

Functions: quiz 6

How many different variables named *i* are created when executing *java-introcs Polynomial 5 2*?

A. 0

B. 1

C. 2

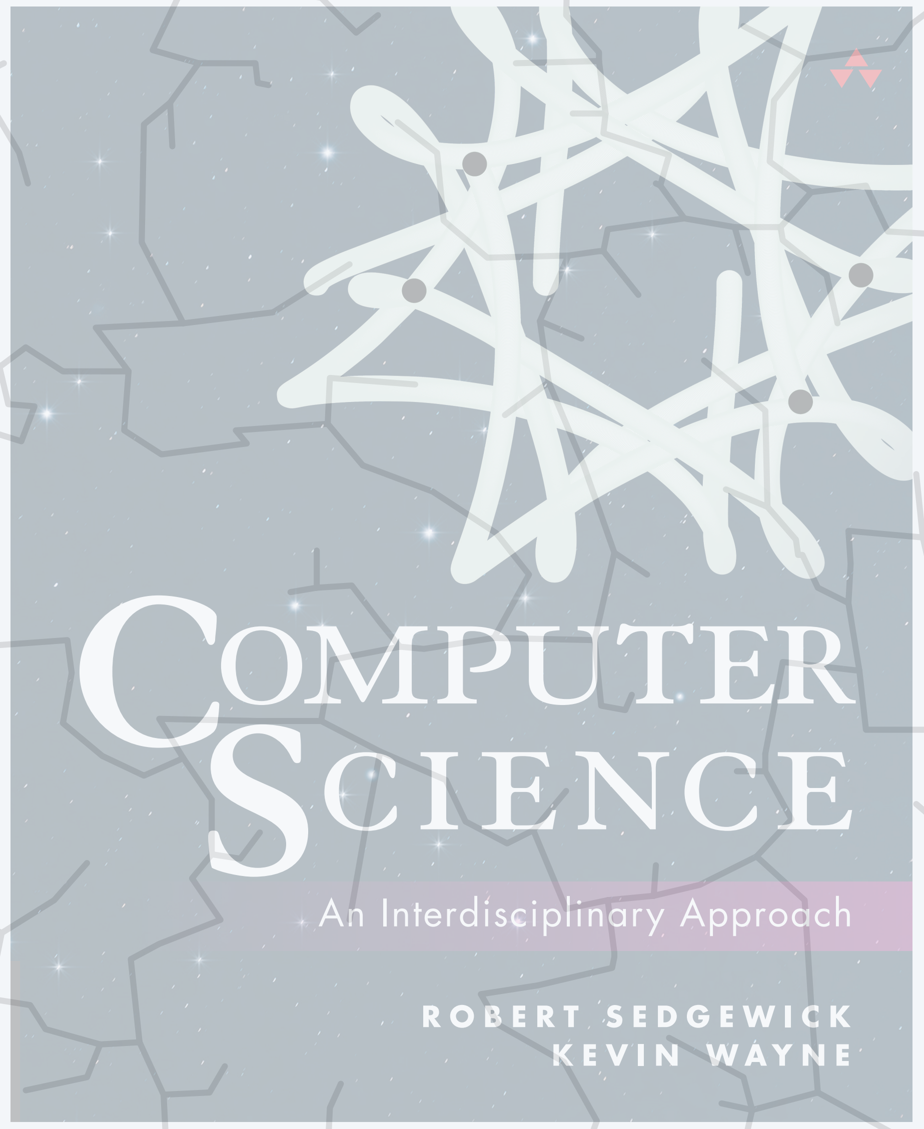
D. 3

E. 4

```
public class Polynomial {

  public static int evaluate(int[] a, int x) {
     int result = 0, monomial = 1;
        for (int i = a.length - 1; i >= 0; i--, monomial *= x)
           result += a[i] * monomial;
     return result;
  }

  public static void main(String[] args) {
     for (int i = 0; i < args.length; i++) {
        int x = Integer.parseInt(args[i]);
        StdOut.println(evaluate(new int[] {1, 0}, x));
     }
  }
}
```


https://introcs.cs.princeton.edu

2.1 Functions

- *flow-of-control*
- *properties*
- *call stack and scope*
- *APIs and libraries*

Application Programming Interface

API. An interface between provider and client programs. $\longleftarrow$ *"contract" between software provider and other software*

Examples.

Application Programming Interface

API. An interface between provider and client programs. ← *"contract" between software provider and other software*

This course. Concise description of the functions available to the client.

Examples.

`public class StdIn`		description
`static boolean`	`isEmpty()`	`true` *if no more values,* `false` *otherwise*
`static int`	`readInt()`	*read a value of type* `int`
`static double`	`readDouble()`	*read a value of type* `double`
`static boolean`	`readBoolean()`	*read a value of type* `boolean`
`static String`	`readString()`	*read a value of type* `String`
	⋮	⋮

Application Programming Interface

API. An interface between provider and client programs. ← *"contract" between software provider and other software*

This course. Concise description of the functions available to the client.

Examples.

`public class StdOut`	**description**
`static void print(String s)`	*print* `s` *on the output stream*
`static void println()`	*print a newline on the output stream*
`static void println(String s)`	*print* `s`*, then a newline on the stream*
`static void printf(String f, ...)`	*print formatted output*
⋮	⋮

Application Programming Interface

API. An interface between provider and client programs. ← *"contract" between software provider and other software*

This course. Concise description of the functions available to the client.

Examples.

public class StdMidi		description
static void	play()	*plays the specified MIDI file*
static void	setInstrument()	*sets the MIDI instrument to the specified value*
static void	setTempo()	*sets the tempo to the specified number of beats per minute*
static void	playNote()	*plays the specified note for the given duration (measured in beats)*
static void	noteOn()	*turns the specified note on*
⋮	⋮	⋮

Application Programming Interface

API. An interface between provider and client programs. ← *"contract" between software provider and other software*

This course. Concise description of the functions available to the client.

Examples.

public class Synth	
`static int`	`length(double duration)`
`static double`	`sine(double frequency, double t)`
`static double`	`square(double frequency, double t)`
`static double`	`saw(double frequency, double t)`
`static double[]`	`sineWave(double frequency, double amplitude, duble duration)`
`static double[]`	`squareWave(double frequency, double amplitude, duble duration)`
`static double[]`	`sawWave(double frequency, double amplitude, duble duration)`
`static double[]`	`whiteNoise(double amplitude, duble duration)`
`static double[]`	`add(double[] a, double[] b)`
`static double[]`	`multiply(double[] a, double[] b)`
`static double[]`	`fade(double[] a, double lambda)`
`static void`	`main(String[] args)` ← `main()` *not called by the client; use for unit testing!*

A *Polynomial* library

Goal. Provide useful operations on non-zero polynomials.

public class Polynomial		description
static int	eval(int[] a, int x)	*evaluate polynomial with coefficients a[] on x*
static double	eval(double[] a, double x)	*evaluate polynomial with coefficients a[] on x*
static void	print(int[] a)	*print polynomial with coefficients a[]*
static void	print(double[] a)	*print polynomial with coefficients a[]*
static double	linearRoot(double[] a)	*root of linear polynomial (degree must be 1)*
static double[]	derivative(double[] a)	*derivative of polynomial with coefficients a[]*
static double	nearestRoot(double[] a, double start)	*root obtained by Newton's method at start point*
static double[]	quadraticRoots(double[] a)	*all roots of quadratic polynomial (degree must be 2)*
static double[]	cubicRoots(double[] a)	*all roots of cubic polynomial (degree must be 3)*
static double[]	quarticRoots(double[] a)	*all roots of quartic polynomial (degree must be 4)*
⋮	⋮	⋮
static void	main(String[] args)	*unit testing*

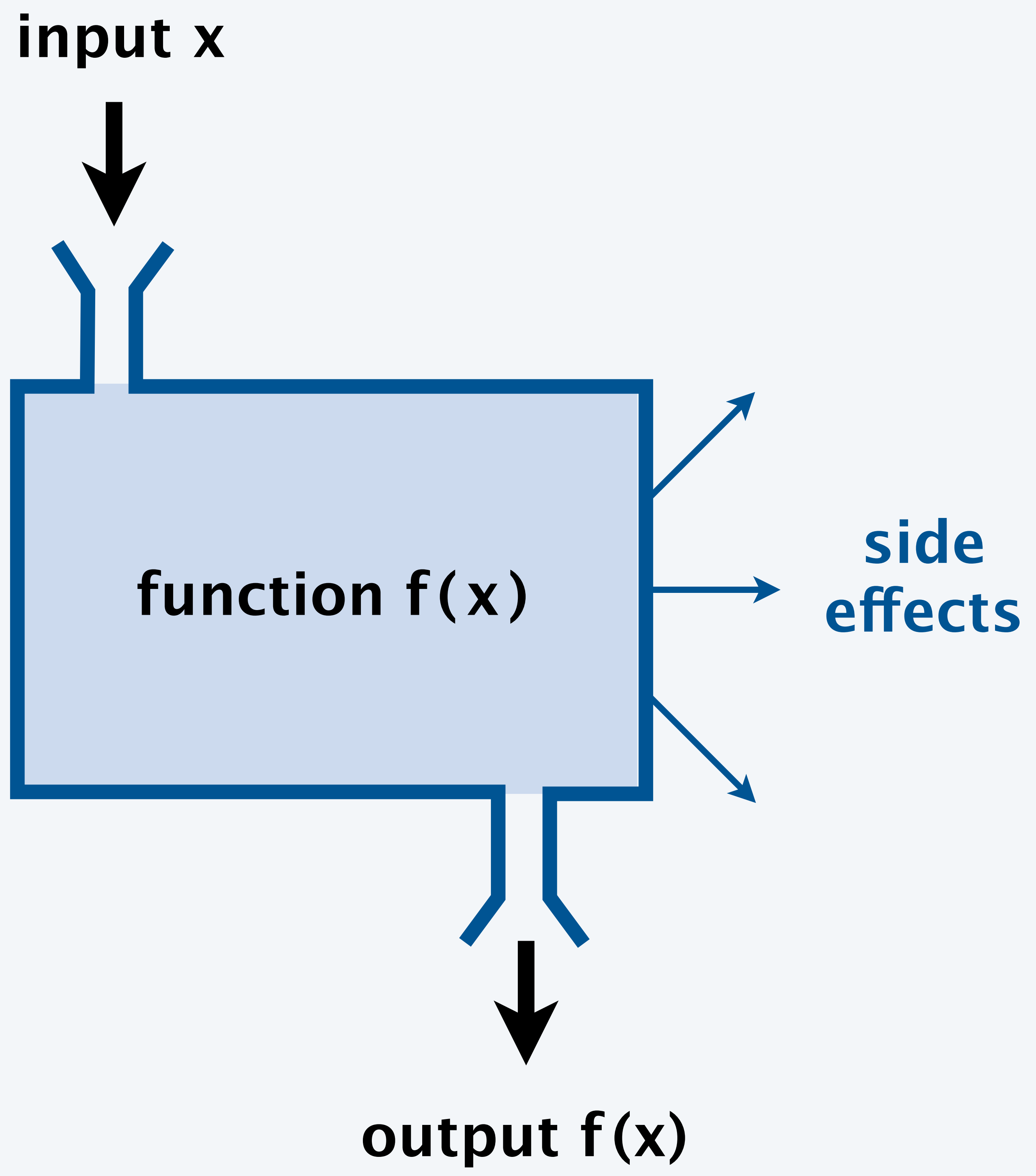
Summary

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

- Java evaluates the arguments and passes by value to function. ← *stay tuned!*
- 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.



Credits

media	source	license
Gears	Adobe Stock	education license
Function Gradient	Adobe Stock	education license
Function Machine	Wvbailey	public domain
Chemotherapy Side Effects	Adobe Stock	education license
Google Maps logo	Wikipedia	public domain
OpenAI logo	Wikipedia	public domain
WhatsApp logo	Wikipedia	public domain
Instagram logo	Wikipedia	public domain