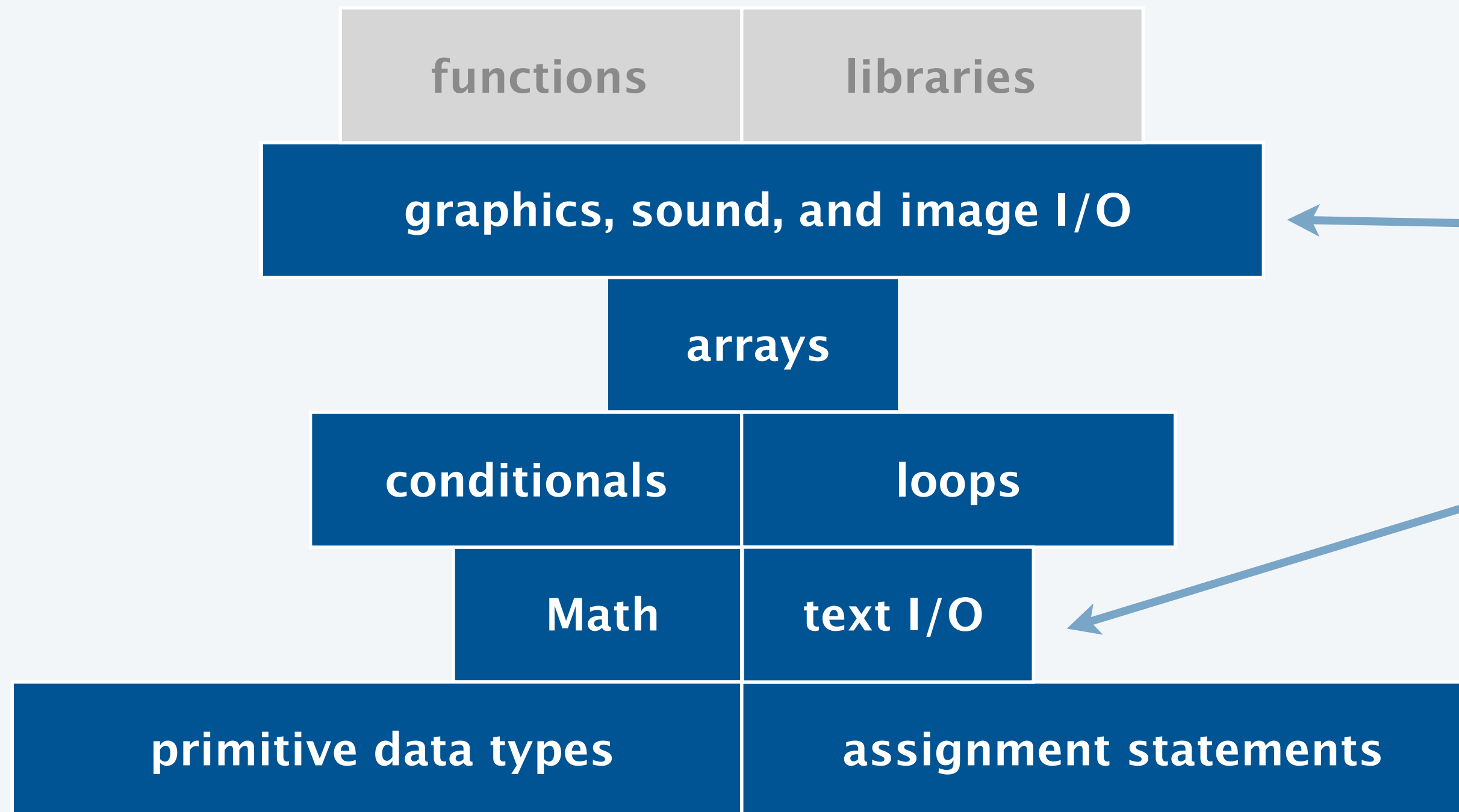


<https://introc.cs.princeton.edu>

1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard MIDI*

Basic building blocks for programming



interact with the outside world

Input and output

Goal. Write Java programs that interact with the outside world via input and output devices.

Input devices.



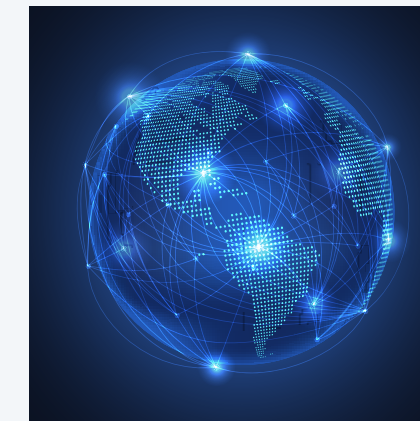
keyboard



trackpad



storage



network



webcam



microphone

Output devices.



video display



earbuds



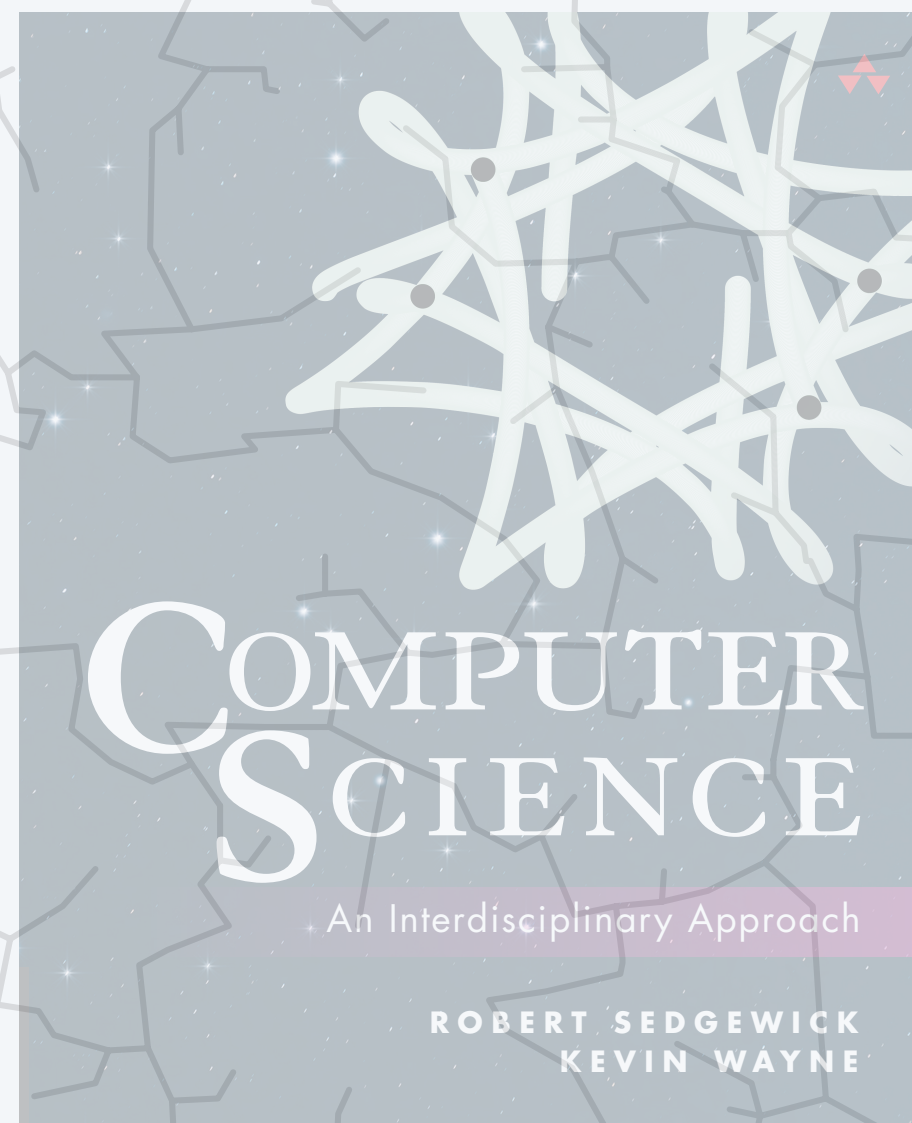
storage



network



braille display



<https://introcs.cs.princeton.edu>

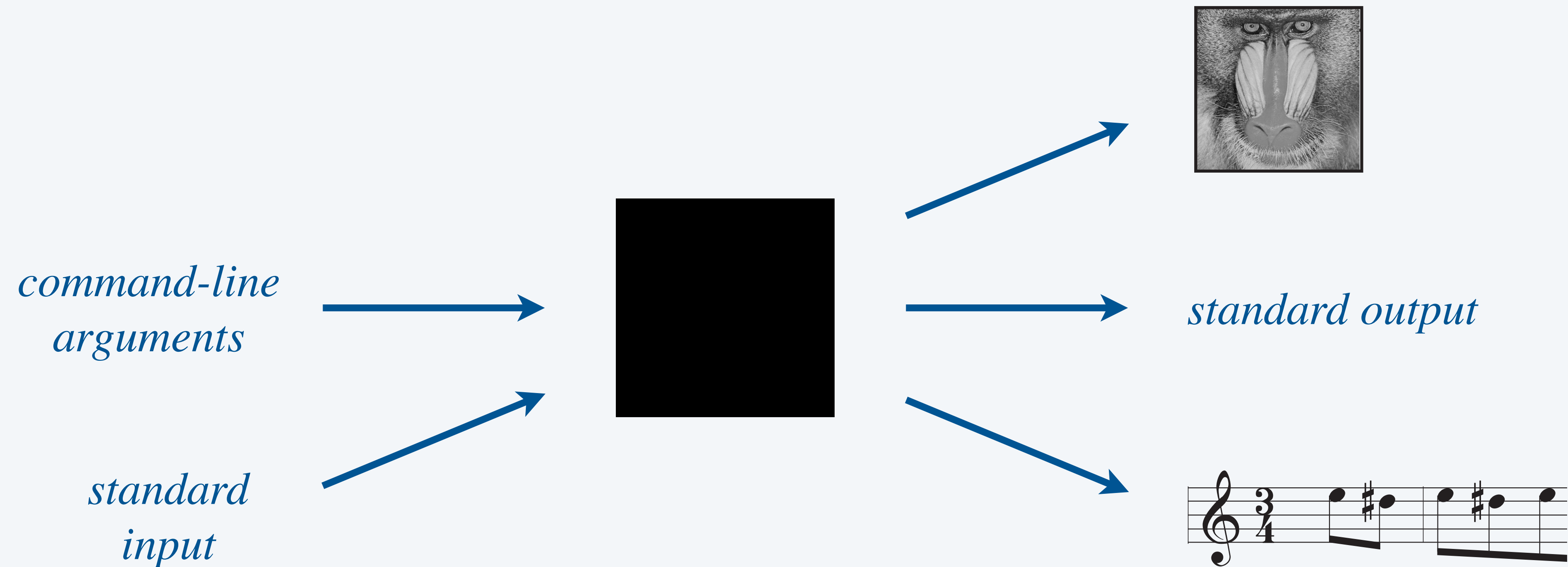
1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard MIDI*

Input-output abstractions (so far)

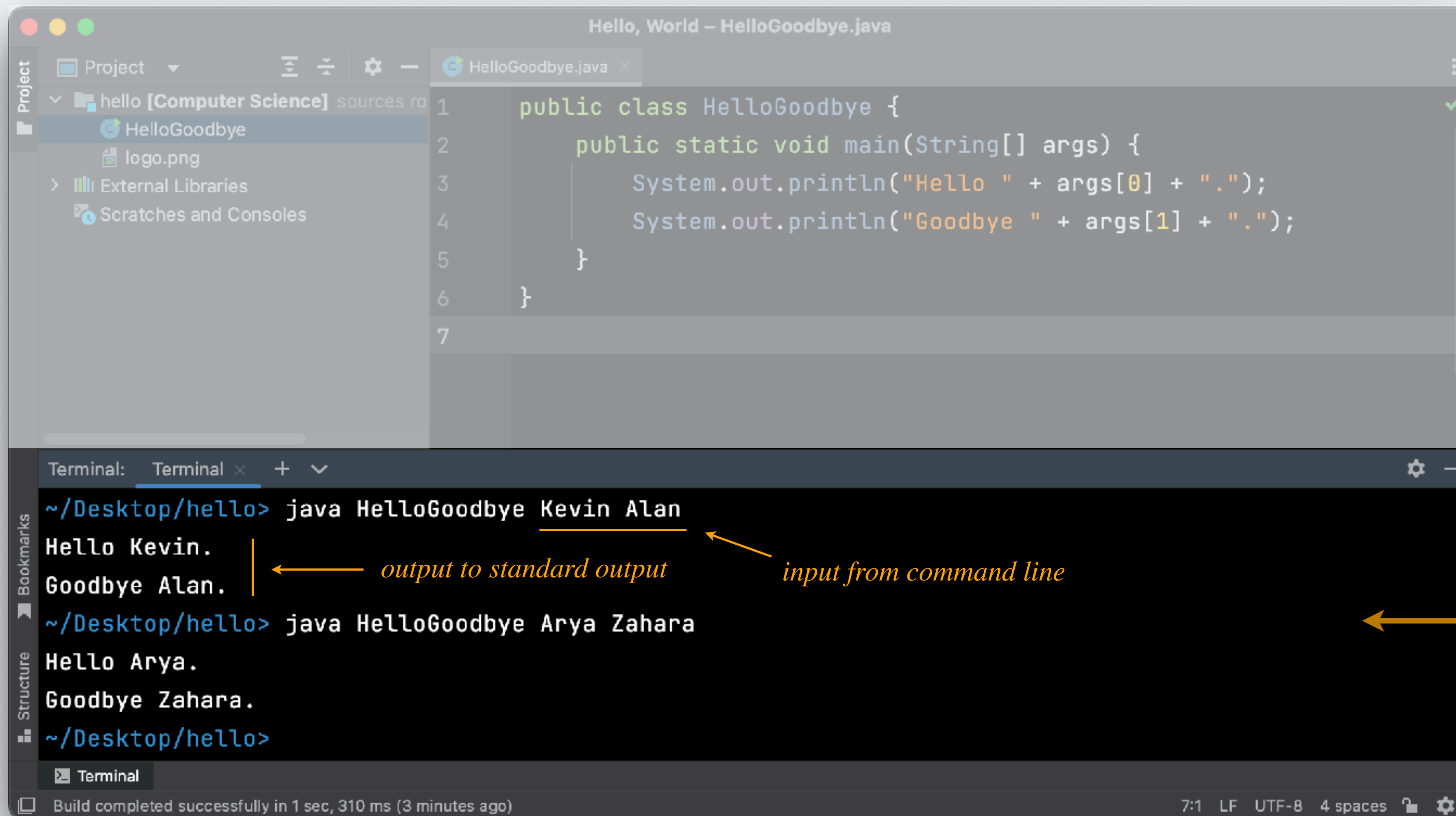
Our approach.

- Define input and output abstractions.
- Use operating system (OS) functionality to connect our Java programs to physical devices.



Review: terminal

Terminal. A text-based interface for interacting with programs, files, and devices.



The screenshot shows an IDE window titled "Hello, World - HelloGoodbye.java". The code editor contains the following Java code:

```
1 public class HelloGoodbye {
2     public static void main(String[] args) {
3         System.out.println("Hello " + args[0] + ".");
4         System.out.println("Goodbye " + args[1] + ".");
5     }
6 }
7
```

Below the code editor is a terminal window. The terminal shows the following commands and output:

```
~/Desktop/hello> java HelloGoodbye Kevin Alan
Hello Kevin.
Goodbye Alan.
~/Desktop/hello> java HelloGoodbye Arya Zahara
Hello Arya.
Goodbye Zahara.
~/Desktop/hello>
```

Annotations in the terminal window:

- An arrow points from the text "output to standard output" to the output lines "Hello Kevin." and "Goodbye Alan.".
- An arrow points from the text "input from command line" to the command "java HelloGoodbye Kevin Alan".

At the bottom of the terminal window, there is a status bar that reads: "Build completed successfully in 1 sec, 310 ms (3 minutes ago) 7:1 LF UTF-8 4 spaces".



VT-100 terminal emulator

Review: command-line arguments

Command-line arguments. Provide text input to a program.

Basic properties.

- Arguments provided to a program by typing after program name.
- Arguments provided to program *before* execution.
- Java: string arguments available in *main()* as *args[0]*, *args[1]*, ...

```
public class HelloGoodbye {  
    public static void main(String[] args) {  
        StdOut.print("Hello ");  
        StdOut.println(args[0] + ".");  
        StdOut.print("Goodbye ");  
        StdOut.println(args[1] + ".");  
    }  
}
```

replaces System.out.println()

```
~/cos126/io> java HelloGoodbye Kevin Alan  
Hello Kevin.  
Goodbye Alan.  
                                     ↑  
                                 command-line arguments
```

```
~/cos126/io> java HelloGoodbye Arya Zahara  
Hello Arya.  
Goodbye Zahara.  
                ↑           ↑  
            args[0]       args[1]
```

```
~/cos126/io> java HelloGoodbye Aðalbjörg "Hua Fei"  
Hello Aðalbjörg.  
Goodbye Hua Fei.  
                                     ↑  
                                 use quotes
```

Review: standard output

Standard output stream. An abstraction for an output sequence of text.

Basic properties.

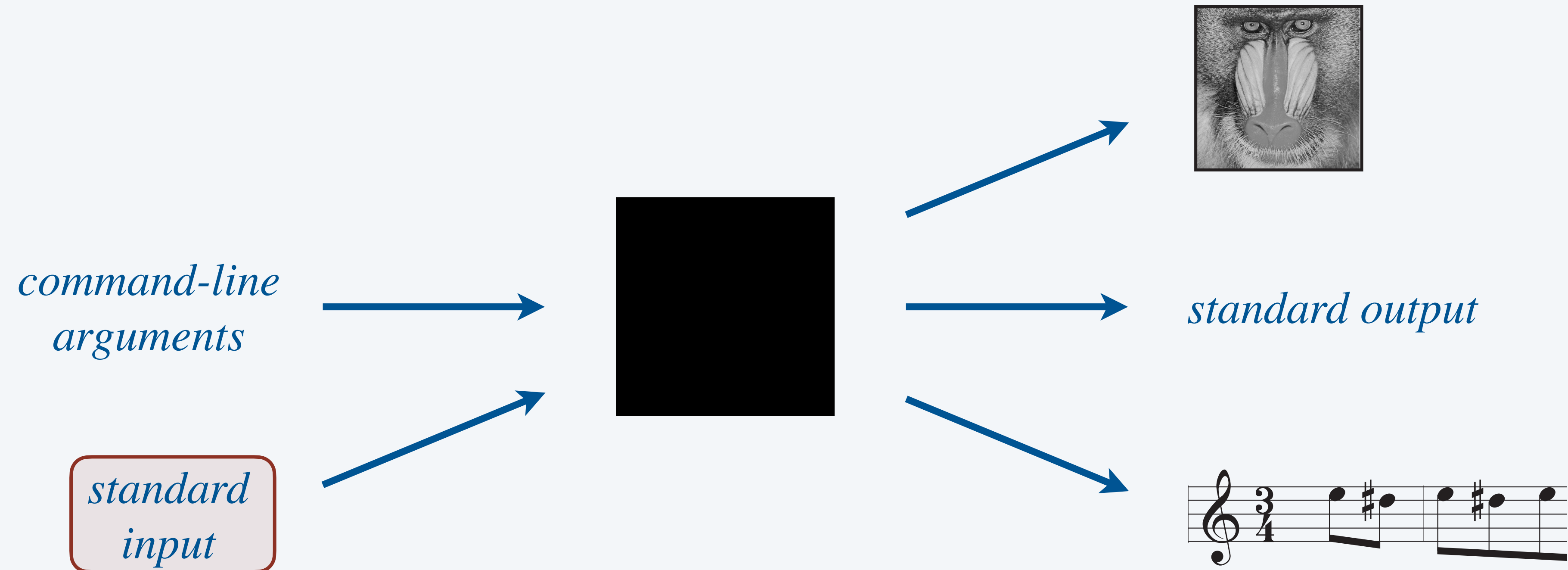
- The call `System.out.println()/StdOut.println()` appends text to the standard output stream.
- By default, the standard output stream is connected to the terminal.
- No limit on amount of output.

```
public class PrintSquares {  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        int square = 0;  
        for (int i = 0; i < n; i++) {  
            square += 2 * i + 1;  
            StdOut.println(square);  
        }  
    }  
}
```

```
~/cos126/io> java PrintSquares 4  
0  
1  
4  
9  
  
~/cos126/io> java PrintSquares 10000  
0  
1  
4  
9  
16  
...  
← produces lots of output
```


Input-output abstractions: standard input

Next step. Add a text **input** stream.

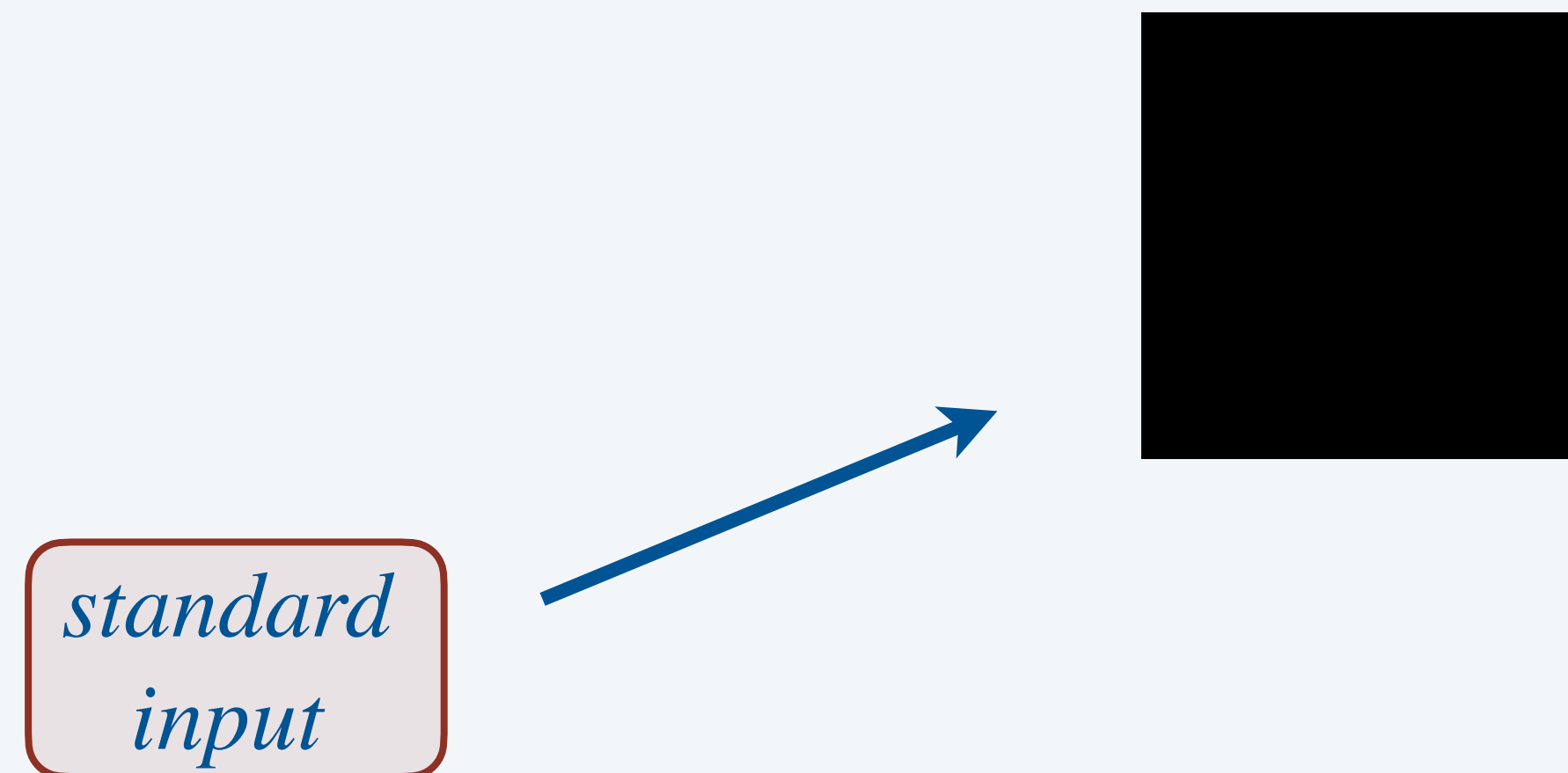


Standard input

Standard input stream. An abstraction for an input sequence of text.

Advantages over command-line arguments:

- No limit on the amount of input.
- Conversion to primitive types is explicitly handled.
- Can provide input interactively, *while* the program is executing.



StdIn. Our library for reading strings and numbers from standard input.

← available with `javac-introcs`
and `java-introcs` commands

<code>public class StdIn</code>	description
<code>static boolean isEmpty()</code>	<i>true if no more values, false otherwise</i>
<code>static int readInt()</code>	<i>read a value of type int</i>
<code>static double readDouble()</code>	<i>read a value of type double</i>
<code>static boolean readBoolean()</code>	<i>read a value of type boolean</i>
<code>static String readString()</code>	<i>read a value of type String</i>
<code>:</code>	<code>:</code>

← reads next token (sequence of non-whitespace characters)
and attempts to parse as specified type

StdOut. Our library for printing strings and numbers to standard output.

available with `javac-introcs`
and `java-introcs` commands

<code>public class StdOut</code>	description
<code>static void print(String s)</code>	<i>print s on the output stream</i>
<code>static void println()</code>	<i>print a newline on the output stream</i>
<code>static void println(String s)</code>	<i>print s, then a newline on the stream</i>
<code>static void printf(String f, ...)</code>	<i>print formatted output</i>
<code>⋮</code>	<code>⋮</code>

Q. How different from `System.out.println()` ?

A. Mostly the same, but output is independent of system and locale. ← *we'll use StdOut from now on*

Twenty Questions

Goal. Find a secret number between 1 and 1,000,000 in twenty guesses (with feedback).

```
public class TwentyQuestions {
    public static void main(String[] args) {
        int secret = (int) Math.random() * 1_000_000;

        StdOut.print("I'm thinking of a number ");
        StdOut.println("between 1 and 1,000,000");
        int guess = 0;

        while (guess != secret) {
            StdOut.print("What's your guess? ");
            guess = StdIn.readInt();

            if (guess < secret) StdOut.println("Too low");
            else if (guess > secret) StdOut.println("Too high");
            else StdOut.println("You win!");
        }
    }
}
```

Add numbers on the standard input stream

Goal. Read a stream of numbers (from standard input) and print their sum (to standard output).

```
public class Sum {  
    public static void main(String[] args) {  
        double sum = 0.0;  
        while (!StdIn.isEmpty())  
            sum += StdIn.readDouble();  
        StdOut.println(sum);  
    }  
}
```

```
~/cos126/io> java-introcs Sum  
1.0  
2.0  
4.0  
2.0  
<Ctrl-D> ← signifies end of standard input  
9.0          (<Ctrl-Z><Enter> on Windows)  
  
~/cos126/io> java-introcs Sum  
10.0 5.0 6.0 3.0  
7.0      32.0 ← values separated  
          by whitespace  
<Ctrl-D>  
63.0
```

Remark. No limit on amount of input. ← “streaming algorithm”
(avoids storing data)

Square every number

Goal. Read a stream of numbers and print their squares.

```
public class SquareAll {  
    public static void main(String[] args) {  
        while (!StdIn.isEmpty()) {  
            double x = StdIn.readDouble();  
            StdOut.println(x * x);  
        }  
    }  
}
```

```
~/cos126/io> java-introcs SquareAll  
1.0  
1.0  
2.0  
4.0  
4.0  
16.0  
<Ctrl-D> ← signifies end of standard input  
                  (<Ctrl-Z><Enter> on Windows)
```

Power every number

Goal. Read n from command line and print the numbers from stream to the power n .

```
public class RaiseAll {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        while (!StdIn.isEmpty()) {
            double x = StdIn.readDouble();
            StdOut.println(Math.pow(x, n));
        }
    }
}
```

```
~/cos126/io> java-introcs RaiseAll 3
1.0
1.0
2.0
8.0
4.0
64.0
<Ctrl-D> ← signifies end of standard input
              (<Ctrl-Z><Enter> on Windows)
```


Divide every number

Goal. Read n from the command line, a of stream numbers and print their ratio by n .

```
public class DivideAll {  
    public static void main(String[] args) {  
        int n = Double.parseDouble(args[0]);  
  
        while (!StdIn.isEmpty())  
            StdOut.println(StdIn.readDouble() / n);  
    }  
}
```

```
~/cos126/io> java-introcs DivideAll 10  
1.0  
0.1  
2.0  
0.2  
4.0  
0.4  
<Ctrl-D> ← signifies end of standard input  
              (<Ctrl-Z><Enter> on Windows)
```

Average

Goal. Read a stream of numbers (from standard input) and print their **average** (to standard output).

```
public class Average {
    public static void main(String[] args) {
        double sum = 0.0;
        int n = 0;

        while (!StdIn.isEmpty()) {
            double x = StdIn.readDouble();
            sum += x;
            n++;
        }

        StdOut.println(sum / n);
    }
}
```

```
~/cos126/io> java-introcs Average
1.0
2.0
4.0
2.0
<Ctrl-D> ← signifies end of standard input
              (<Ctrl-Z><Enter> on Windows)
2.25

~/cos126/io> java-introcs Average
10.0 5.0 6.0 3.0
7.0      32.0 ← values separated
                by whitespace
<Ctrl-D>
10.5
```



What does the following program do with the given input?

- A. Prints "A", "B", and "C".
- B. Throws an error.
- C. Both A and B.
- D. Neither A nor B.

```
public class Mystery {  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        for (int i = 0; i < n; i++) {  
            String s = StdIn.readString();  
            StdOut.println(s);  
        }  
    }  
}
```

```
~/cos126/io> java-introcs Mystery 5  
A B C  
<Ctrl-D>
```

The printf() method

Print with formatting. Choose number of characters and precision.

```
public class LotsOfPrints {
    public static void main(String[] args) {
        StdOut.printf("An integer: %5d\n", 10);
        StdOut.printf("A double: %5.2f\n", 10.255);
        StdOut.printf("Another double: %5.2f\n", 100.255);
        StdOut.printf("%d + %d = %d\n", 2, 2, 4);
        StdOut.printf("A string: %10s\n", "blah");
        StdOut.printf("pi is approximately %.20f\n", Math.PI);
    }
}
```

```
~/cos126/io> java-introcs LotsOfPrints
An integer:      10
A double: 10.26
Another double: 100.26
2 + 2 = 4
A string:          blah
pi is approximately 3.14159265358979300000000000000000
```

Remark 1. Needs `\n` (newline character) to go to next line.

Remark 2. In `%x.yf`, `x` is a floor: if number takes more, same as `%.yf`. (Likewise for `%s`, `%d`, etc.)



What does the following program print?

- A. "1.2 / 1.2 = 1.0"
- B. "1.25 / 1.25 = 1.0"
- C. "1.3 / 1.25 = 1.0"
- D. Throws an error.

```
public class AnotherMystery {  
    public static void main(String[] args) {  
        double a = Double.parseDouble(args[0]);  
        double b = Double.parseDouble(args[1]);  
        StdOut.printf("%.1f / %.2f = %.1f\n", a, b, a / b);  
    }  
}
```

```
~/cos126/io> java-introcs AnotherMystery 1.25 1.25
```



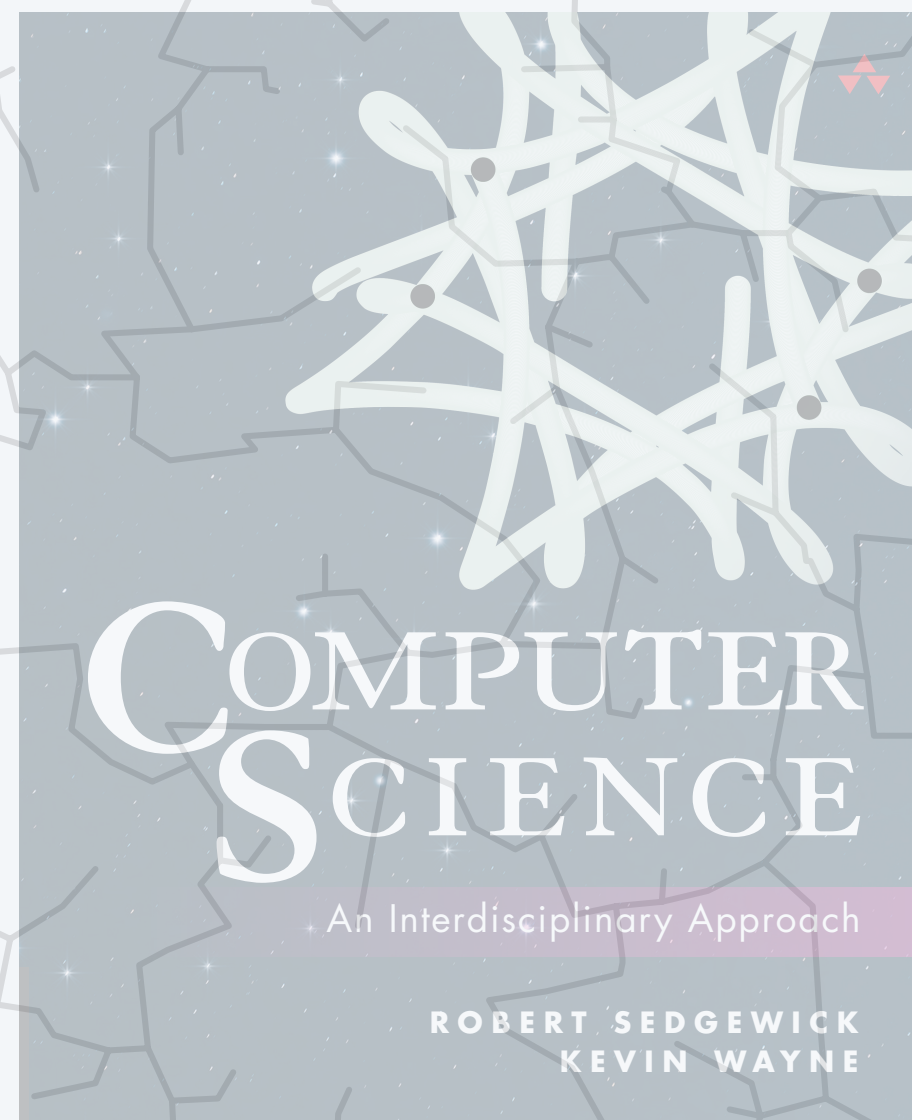
What does the following program print?

- A. "1.25 is larger than 1.20 by 4.1666667%"
- B. "1.3 is larger than 1.2 by 4.1666667%"
- C. "1.20 = 1.25"
- D. Throws an error.

```
~/cos126/io> java-introcs YetAnotherMystery 1.2 1.25
```

```
public class YetAnotherMystery {
    public static void main(String[] args) {
        double a = Double.parseDouble(args[0]);
        double b = Double.parseDouble(args[1]);

        if (a > b) StdOut.printf("%.2f is larger than %.2f by %f\n", a, b, 100 * (a / b - 1.0));
        else if (b > a) StdOut.printf("%.1f is larger than %.1f by %f", b, a, 100 * (b / a - 1.0));
        else StdOut.printf("%.2f = %.2f", a, b);
    }
}
```



<https://introcs.cs.princeton.edu>

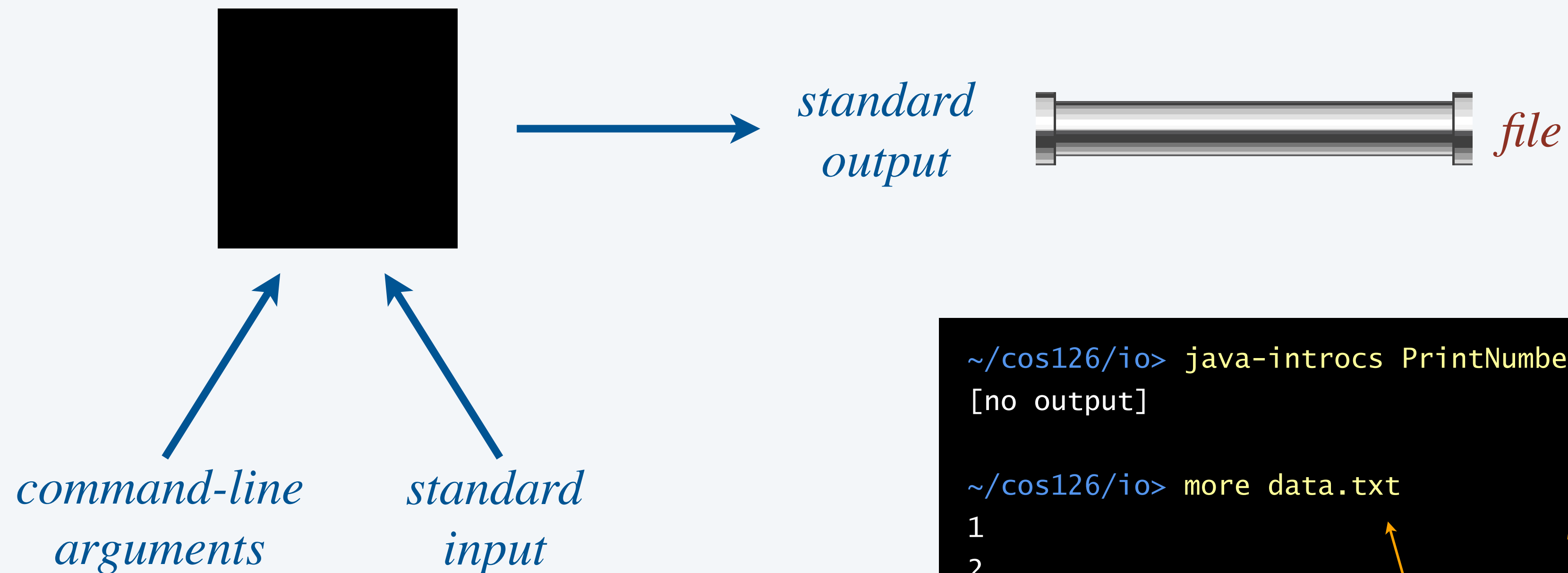
1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard MIDI*

Redirecting standard output

Terminal. By default, standard output is connected to the terminal.

Redirecting standard output. Send standard output to a **file** (instead of the terminal).



```
~/cos126/io> java-introcs PrintNumbers 100 > data.txt
[no output]

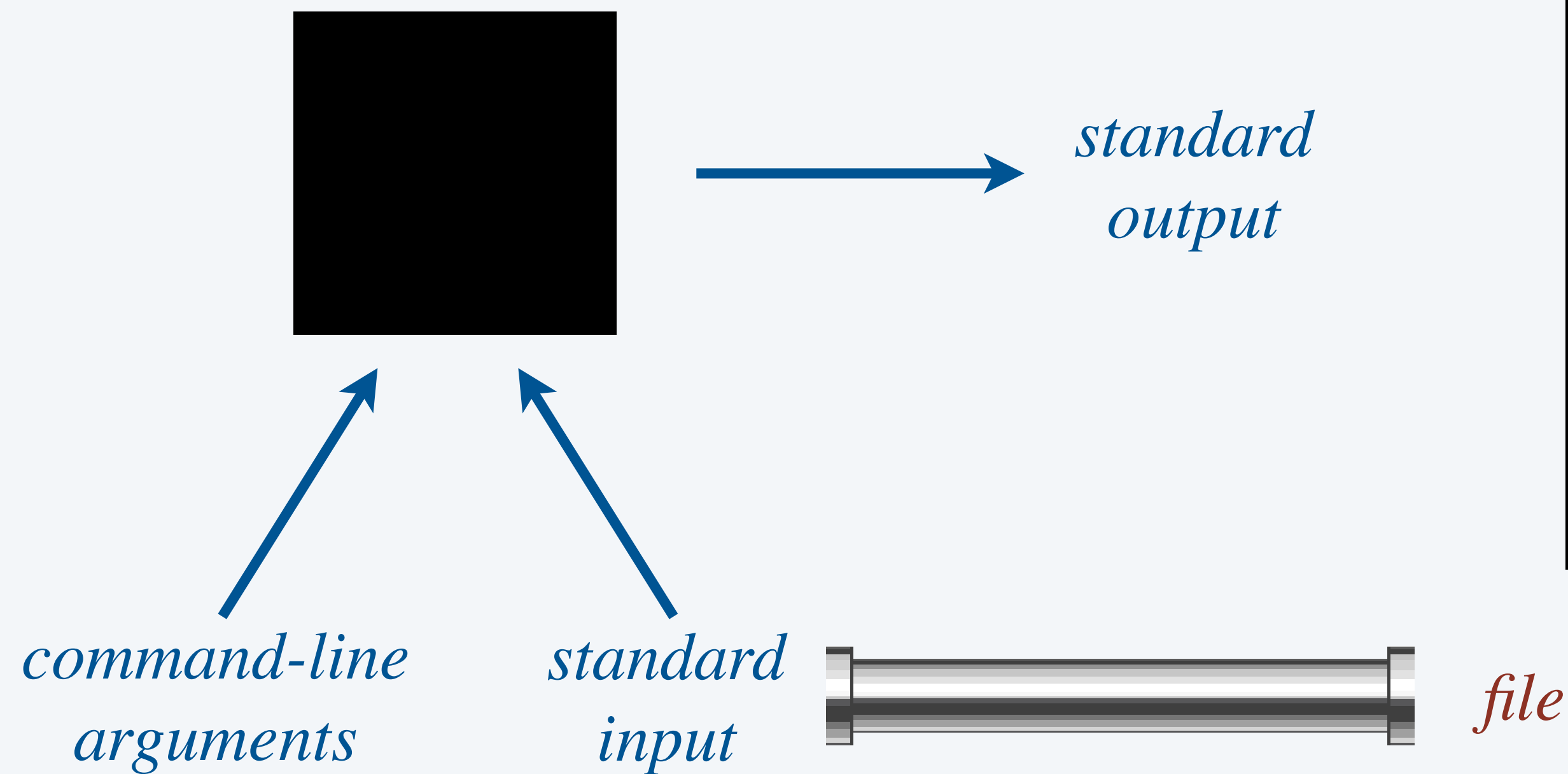
~/cos126/io> more data.txt
1
2
3
4
5
...
```

redirect standard output (arrow pointing to `>`)
filename (arrow pointing to `data.txt`)
display content of a file (arrow pointing to `more data.txt`)

Redirecting standard input

Terminal. By default, standard input is connected to the terminal.

Redirecting standard input. Read standard input from a **file** (instead of the terminal).



```
~/cos126/io> more data.txt
1
2
3
4
5
...

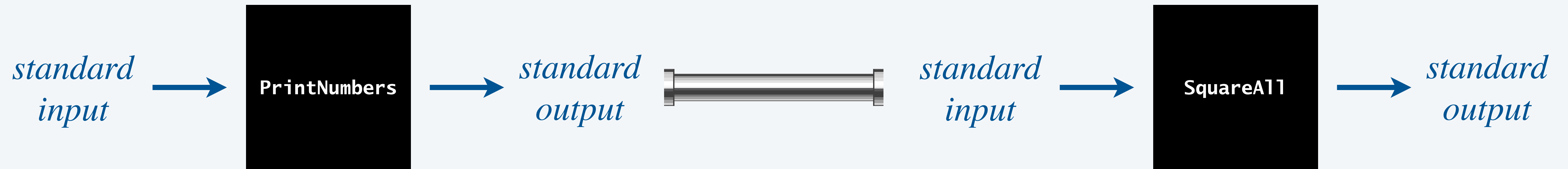
~/cos126/io> java-introcs Sum < data.txt
5050
```

redirect standard input (points to `<`)

filename (points to `data.txt`)

Piping

Piping. Connect standard output of one program to standard input of another program.



```
~/cos126/io> java-introcs PrintNumbers 3 | java-introcs SquareAll
1
4
9
~/cos126/io> java-introcs PrintNumbers 100 | java-introcs Average
50.5
```

pipe operator

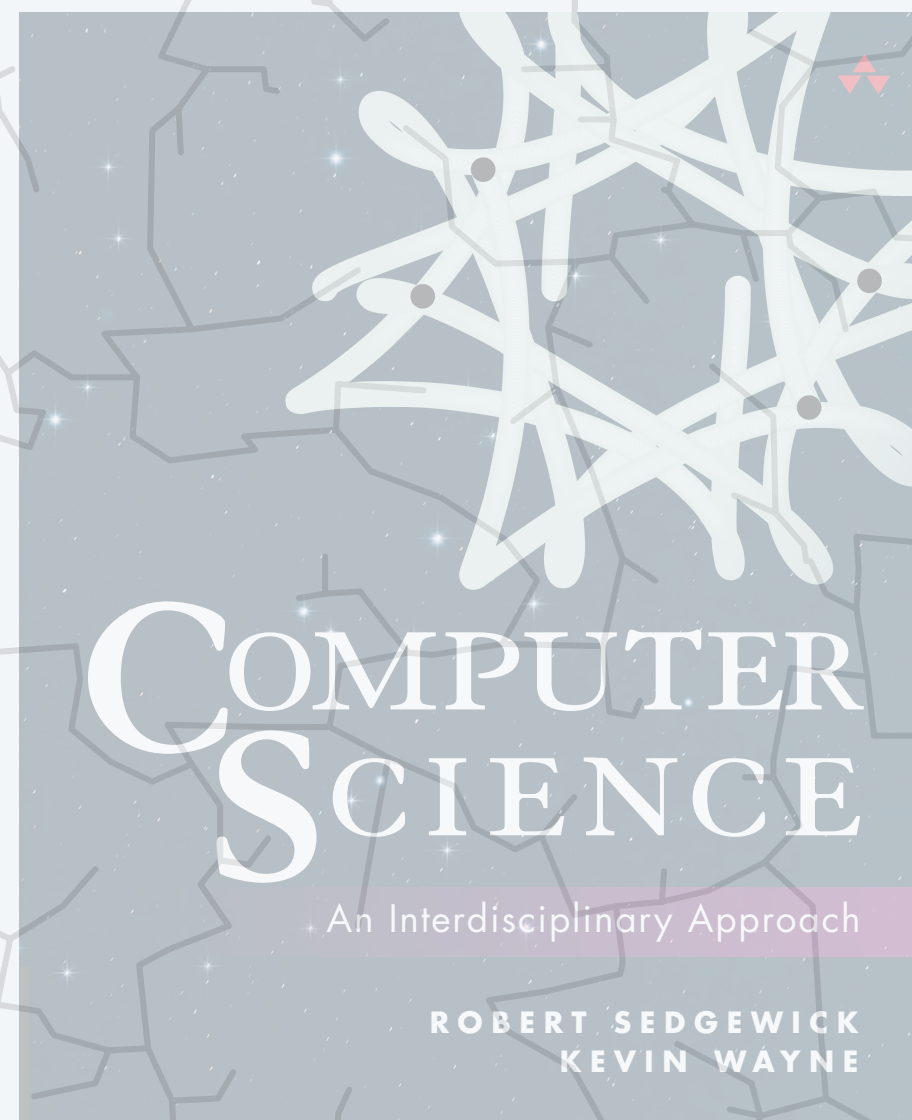
Remark. No limit within programs on amount of data to process.



What is the output of the following command?

```
> java-introcs PrintNumbers 100 | java-introcs DivideAll 100 | java-introcs Sum
```

- A. Integers from 1 to 100.
- B. Squares of integers from 1 to 100.
- C. Ratios by 100 of integers from 1 to 100.
- D. 50.5.
- E. None of the above.



<https://introcs.cs.princeton.edu>

1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard MIDI*

StdMidi. Our library for manipulating music in MIDI format. ← *available with javac-introcs and java-introcs commands*

<code>public class StdMidi</code>	description
<code>static void play()</code>	<i>plays the specified MIDI file</i>
<code>static void setInstrument()</code>	<i>sets the MIDI instrument to the specified value</i>
<code>static void setTempo()</code>	<i>sets the tempo to the specified number of beats per minute</i>
<code>static void playNote()</code>	<i>plays the specified note for the given duration (measured in beats)</i>
<code>static void noteOn()</code>	<i>turns the specified note on</i>
<code>static void pause()</code>	<i>pauses for the specified duration</i>
<code>static void noteOff()</code>	<i>turns specified note off</i>
<code>⋮</code>	<code>⋮</code>

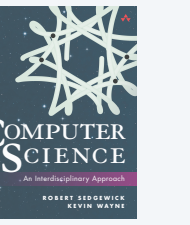
Play MIDI notes

```
public class PlayMidiNotes {  
    public static void main(String[] args) {  
        int instrument = Integer.parseInt(args[0]);  
        StdMidi.setInstrument(instrument);  
  
        while (!StdIn.isEmpty()) {  
            int note = StdIn.readInt();  
            StdMidi.playNote(note);  
        }  
    }  
}
```

```
~/cos126/io> java-introcs PlayMidiNotes 1  
60 62 64 65 67 69 71 72  
<Ctrl-D>
```

```
~/cos126/io> java-introcs PlayMidiNotes 50  
60 62 64 65 65 65 60 62 60 62 62 62 60 67  
65 64 64 64 60 62 64 65 65 65  
<Ctrl-D>
```

Our standard libraries



StdPicture. For manipulating images.

StdAudio. For playing, reading and saving digital audio.

StdIn. For reading strings and numbers from standard input.

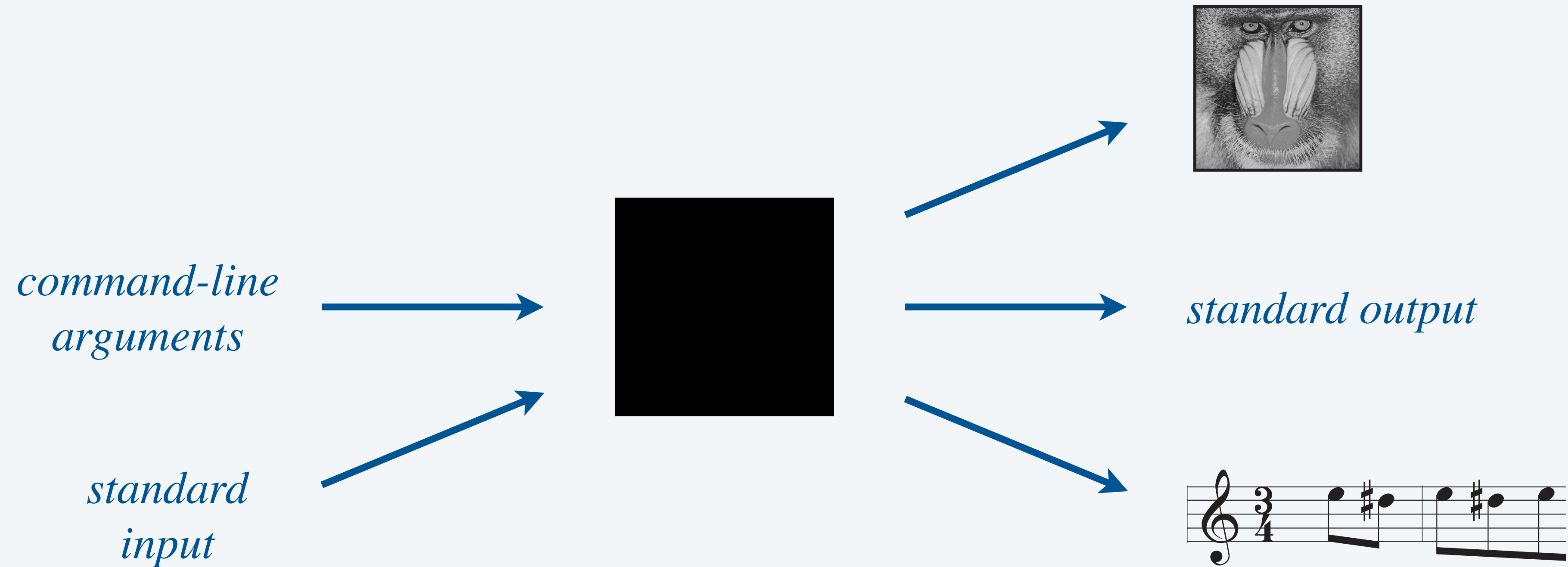
StdOut. For printing strings and numbers to standard output.

StdMidi. For manipulating music in MIDI format.

StdDraw. For creating drawings and animations. ← *not used here, but will be in COS 126!*

Input-output abstractions

Summary. Input and output for text, pictures, drawings, and audio.



Credits

media	source	license
<i>Computer Monitor</i>	<u>iStock</u>	<u>standard license</u>
<i>DEC VT100 Terminal</i>	<u>Wikimedia</u>	<u>CC BY-SA 4.0</u>
<i>Mandrill</i>	<u>USC SIPI Image Database</u>	
<i>Pipe</i>	<u>Adobe Stock</u>	Education License