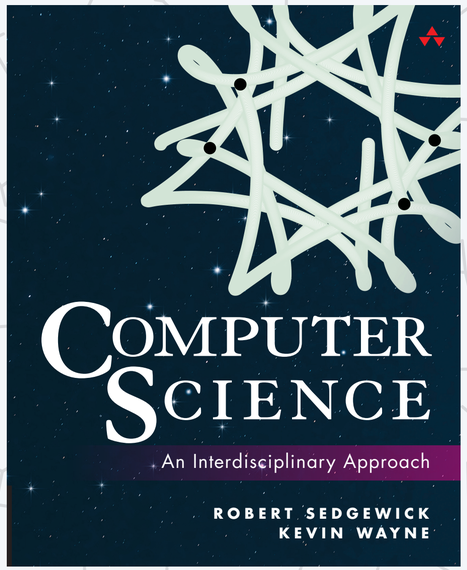
Robert Sedgewick | Kevin Wayne

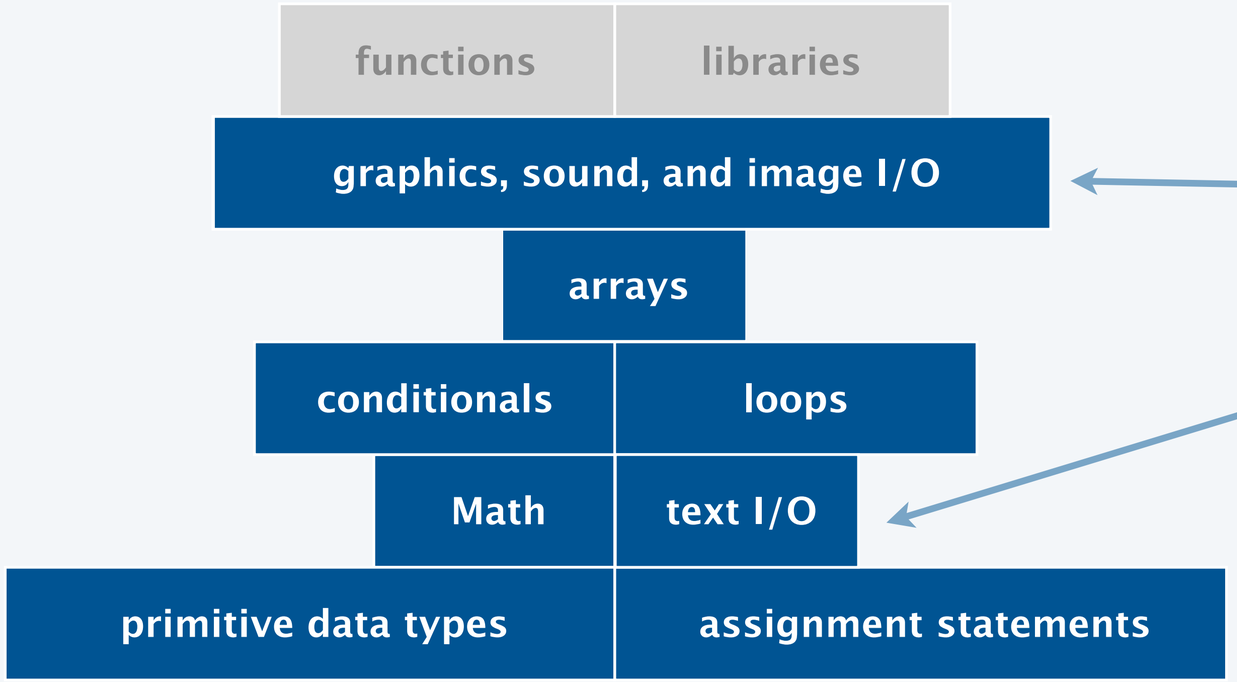


https://introcs.cs.princeton.edu

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.

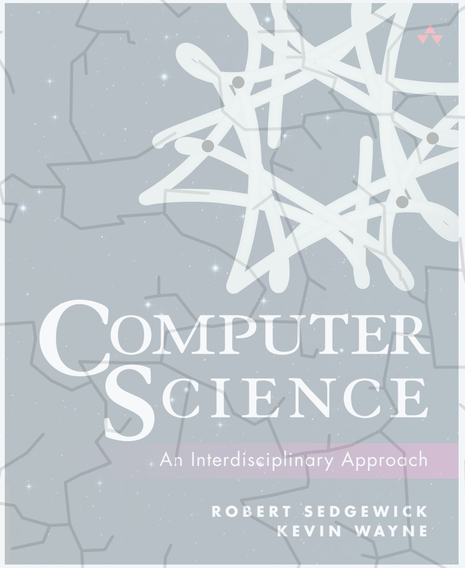
video display

earbuds

storage

network

braille display



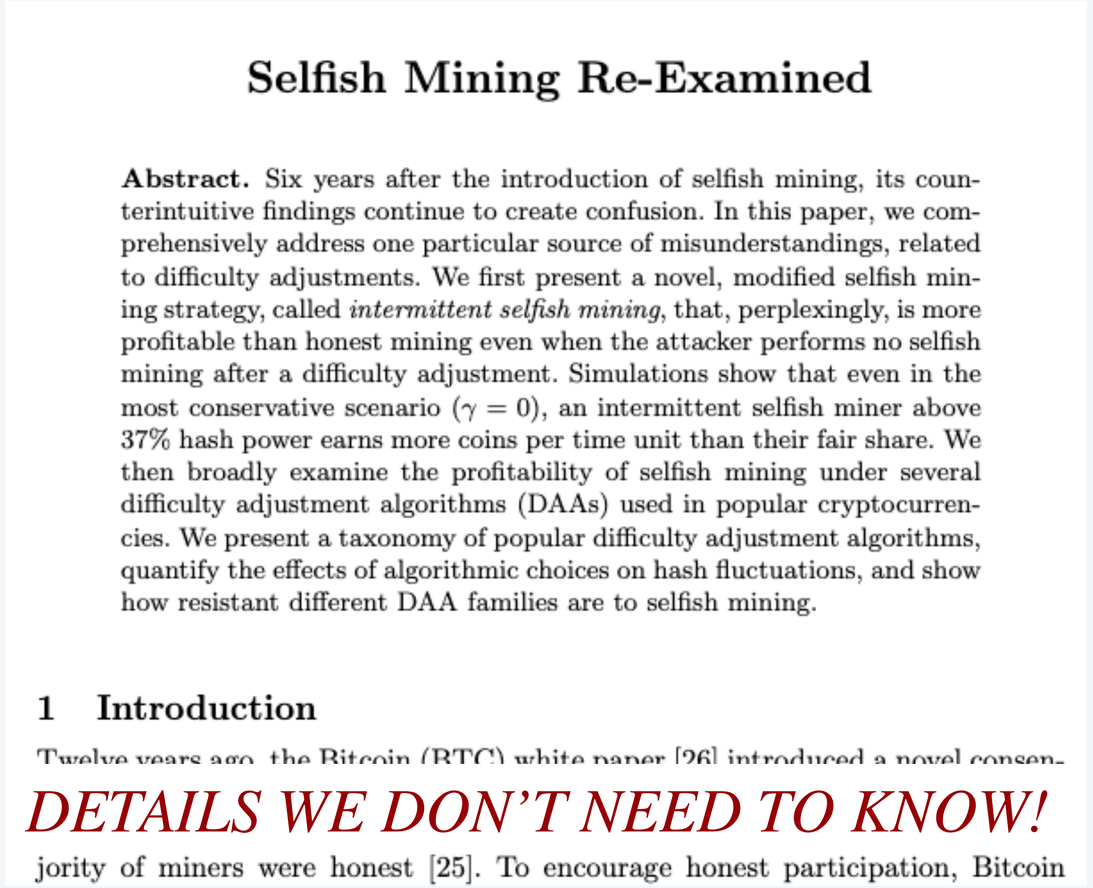
https://introcs.cs.princeton.edu

1.5 Input and Output

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

A note on "abstractions"

- Definitions
  - "Something which exists only as an idea." – COS 126
  - "Disassociated from any specific instance." – Merriam Webster
  - "Generalized so that internal details are hidden away." – Kevin Negy



Selfish Mining Re-Examined

Abstract. Six years after the introduction of selfish mining, its counterintuitive findings continue to create confusion. In this paper, we comprehensively address one particular source of misunderstandings, related to difficulty adjustments. We first present a novel, modified selfish mining strategy, called *intermittent selfish mining*, that, perplexingly, is more profitable than honest mining even when the attacker performs no selfish mining after a difficulty adjustment. Simulations show that even in the most conservative scenario ($\gamma = 0$), an intermittent selfish miner above 37% hash power earns more coins per time unit than their fair share. We then broadly examine the profitability of selfish mining under several difficulty adjustment algorithms (DAAs) used in popular cryptocurrencies. We present a taxonomy of popular difficulty adjustment algorithms, quantify the effects of algorithmic choices on hash fluctuations, and show how resistant different DAA families are to selfish mining.

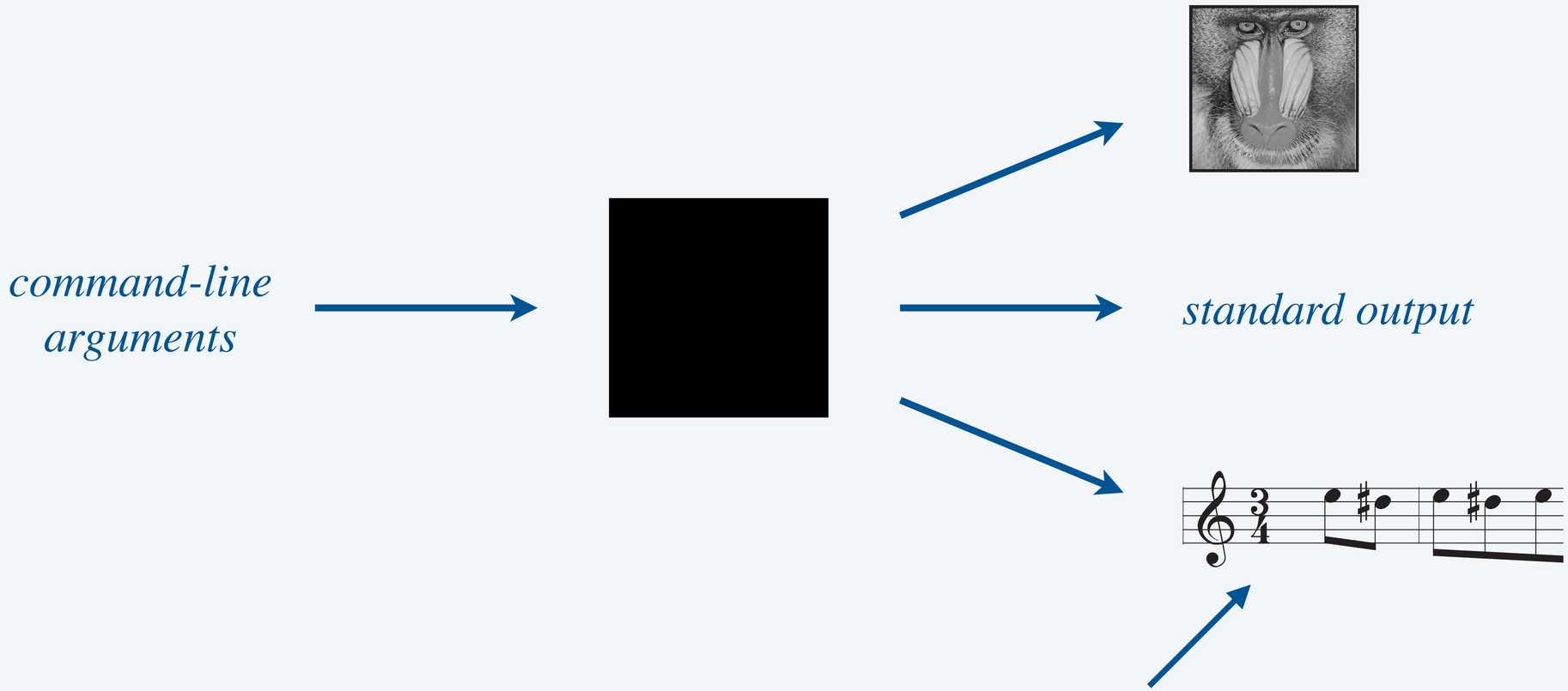
1 Introduction

Twelve years ago, the Bitcoin (BTC) white paper [26] introduced a novel consen-

DETAILS WE DON'T NEED TO KNOW!

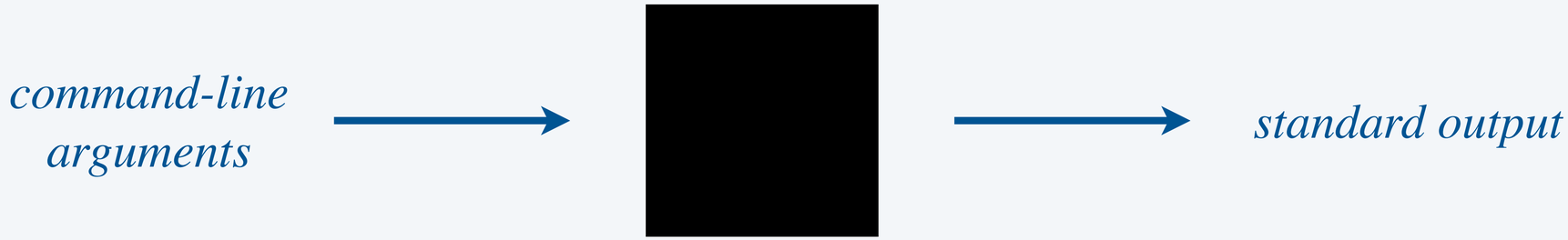
jority of miners were honest [25]. To encourage honest participation, Bitcoin

Input–output abstractions (so far)



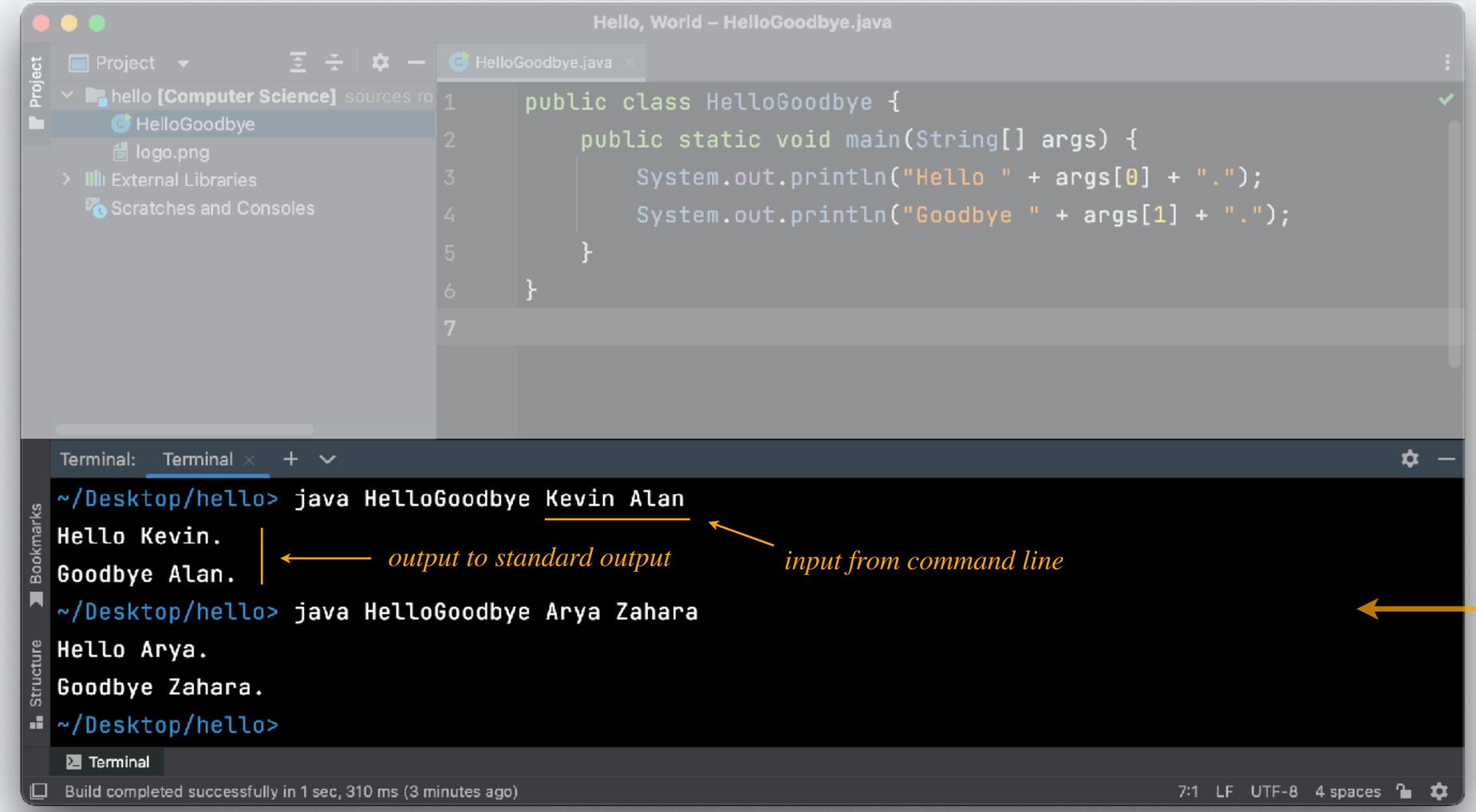
Abstraction – we can use StdMidi.play() without knowing how it works under the hood!

Input–output abstractions (so far)



Review: terminal

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



VT-100 terminal emulator

What are some limitations of our existing text IO methods?

- Non-interactive
- Technically limited list of command line arguments
- What if we want to read input from file?
- What if we want to store output for later?
- What if we want to send output to another application?

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.
- There is a memory limit!

```
kevinnegy@COS-MGXYTF9C72:~$ getconf ARG_MAX
1048576
```

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.
- There is a memory limit!

```
kevinnegy@COS-MGXYTF9C72:~$ getconf ARG_MAX
1048576
kevinnegy@COS-MGXYTF9C72:~$ ls small/
a.java  b.java  c.py
```

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.
- There is a memory limit!

```
kevinnegy@COS-MGXYTF9C72:~$ getconf ARG_MAX
1048576
kevinnegy@COS-MGXYTF9C72:~$ ls small/
a.java  b.java  c.py
kevinnegy@COS-MGXYTF9C72:~$ ls small/a.java small/b.java
small/a.java  small/b.java
```

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.
- There is a memory limit!

```
kevinnegy@COS-MGXYTF9C72:~$ getconf ARG_MAX
1048576
kevinnegy@COS-MGXYTF9C72:~$ ls small/
a.java  b.java  c.py
kevinnegy@COS-MGXYTF9C72:~$ ls small/a.java small/b.java
small/a.java  small/b.java
kevinnegy@COS-MGXYTF9C72:~$ ls small/*.java
```

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.
- There is a memory limit!

```
kevinnegy@COS-MGXYTF9C72:~$ getconf ARG_MAX
1048576
kevinnegy@COS-MGXYTF9C72:~$ ls small/
a.java  b.java  c.py
kevinnegy@COS-MGXYTF9C72:~$ ls small/a.java small/b.java
small/a.java  small/b.java
kevinnegy@COS-MGXYTF9C72:~$ ls small/*.java
small/a.java  small/b.java
```

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.
- There is a memory limit!

```
kevinnegy@COS-MGXYTF9C72:~$ getconf ARG_MAX
1048576
kevinnegy@COS-MGXYTF9C72:~$ ls small/
a.java  b.java  c.py
kevinnegy@COS-MGXYTF9C72:~$ ls small/a.java small/b.java
small/a.java  small/b.java
kevinnegy@COS-MGXYTF9C72:~$ ls small/*.java
small/a.java  small/b.java
kevinnegy@COS-MGXYTF9C72:~$ ls */*/*/*/*
```

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.
- There is a memory limit!

```
kevinnegy@COS-MGXYTF9C72:~$ getconf ARG_MAX
1048576
kevinnegy@COS-MGXYTF9C72:~$ ls small/
a.java  b.java  c.py
kevinnegy@COS-MGXYTF9C72:~$ ls small/a.java small/b.java
small/a.java  small/b.java
kevinnegy@COS-MGXYTF9C72:~$ ls small/*.java
small/a.java  small/b.java
kevinnegy@COS-MGXYTF9C72:~$ ls */*/*/*/*
-bash: /bin/ls: Argument list too long
```

Review: standard output

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

Basic properties.

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

```
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);
      }
   }
```

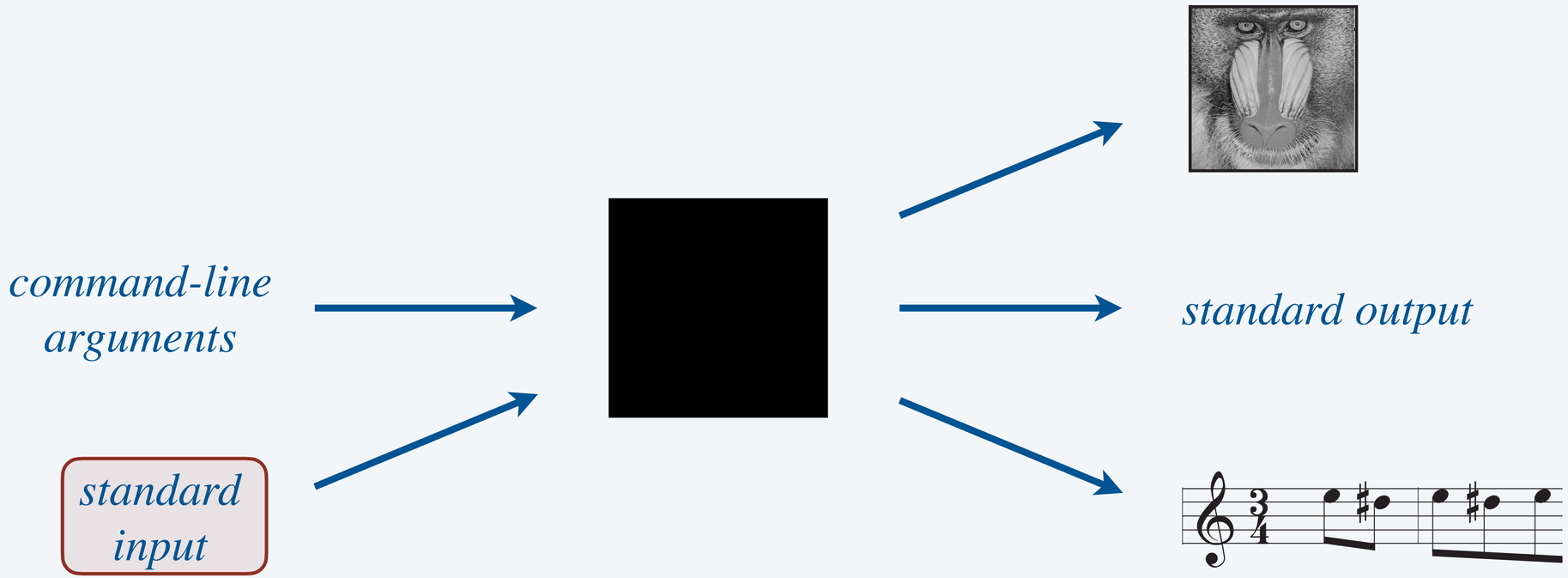
replaces System.out.println()

```
~/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.

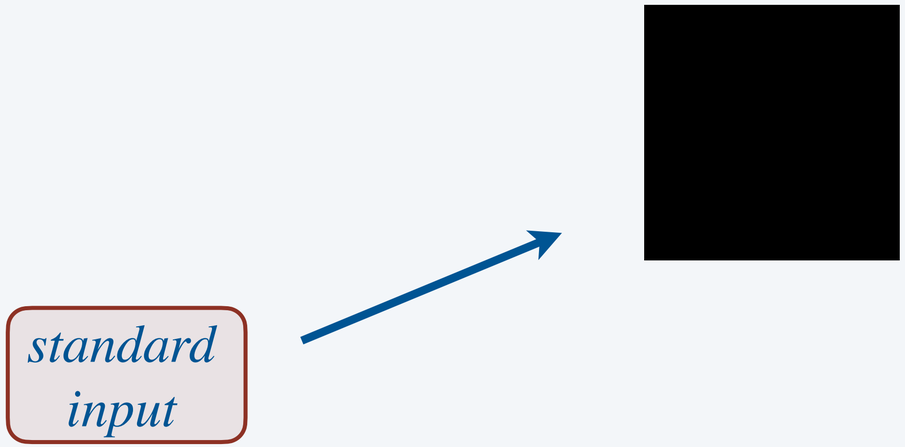


Standard input

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

Advantages over command-line arguments:

- No limit on amount of input.
- Can provide input interactively during execution.
- Conversion to primitive types explicitly handled (with our library `StdIn`).



Standard input library

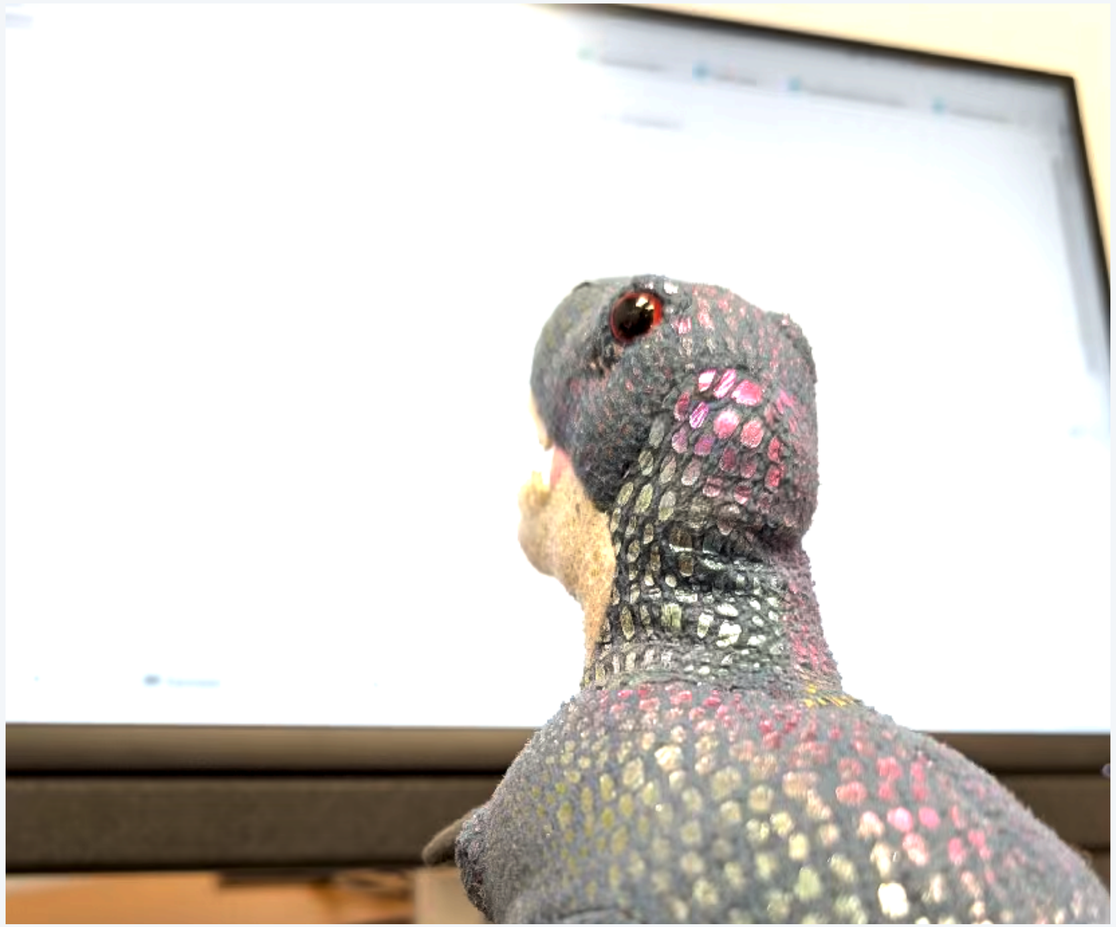
StdIn. Our library for reading strings and numbers from standard input.

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

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
⋮	⋮

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

Live coding



Guess the number

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

```
public class GuessNumber {
   public static void main(String[] args) {
      int secret = (int) (Math.random() * 1000000);

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

      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>
9.0

~/cos126/io> java-introcs Sum
10.0 5.0 6.0 3.0
7.0      32.0
<Ctrl-D>
63.0
```

signifies end of standard input (<Ctrl–Z><Enter> on Windows)

values separated by whitespace

Remark. No limit on amount of input.

"streaming algorithm" (avoids storing data)

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>
2.25

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

signifies end of standard input (<Ctrl–Z><Enter> on Windows)

values separated by whitespace

Input and output: quiz 1

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>
```

Standard output library

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

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

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*
⋮	⋮

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*

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.)

Input and output: quiz 2

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
```

Input and output: quiz 3

What does the following program print?

A. “1.25 is larger than 1.20 by 4.1666667%”

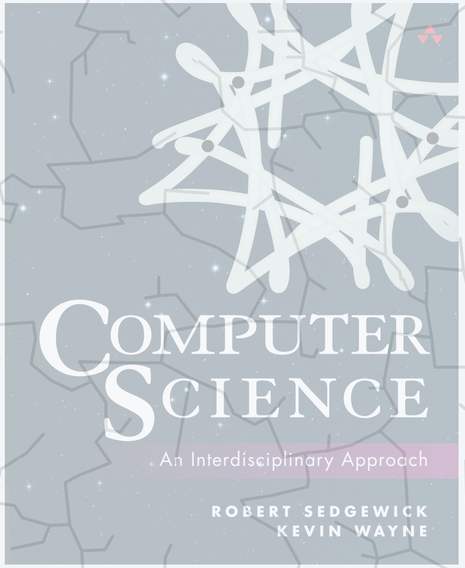
B. “1.3 is larger than 1.2 by 4.1666667%”

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*

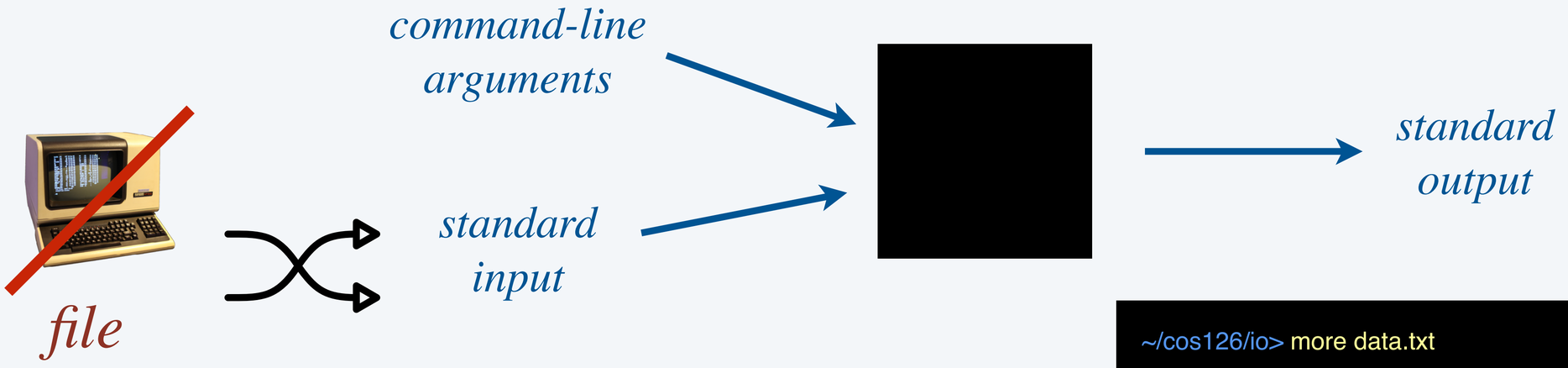
What are some limitations of our existing text IO methods?

- ~~Non-interactive~~
  - StdIn
- ~~Technically limited list of command line arguments~~
  - StdIn
- What if we want to read input from file?
- What if we want to store output for later?
- What if we want to send output to another application?

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).



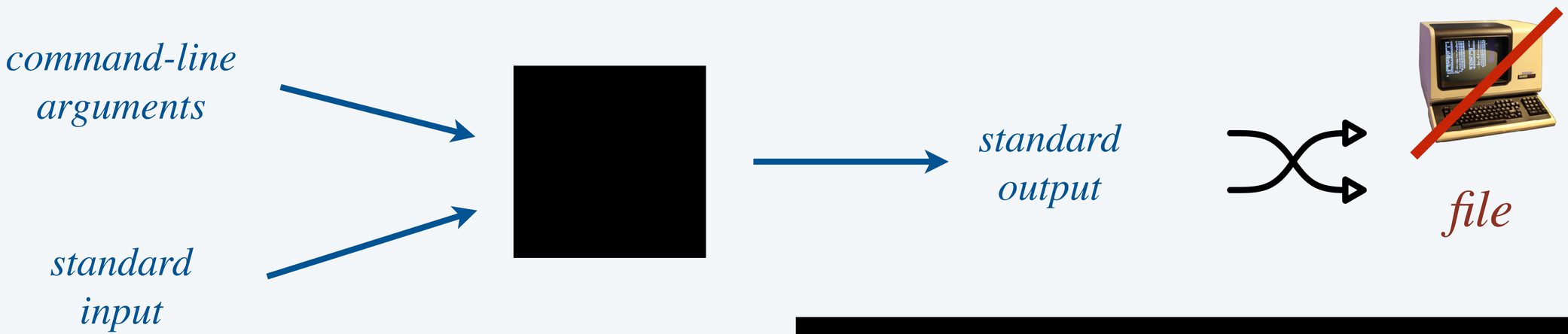

```
~/cos126/io> more data.txt
1
2
3
4
5
...

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

redirect standard input *filename*

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

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

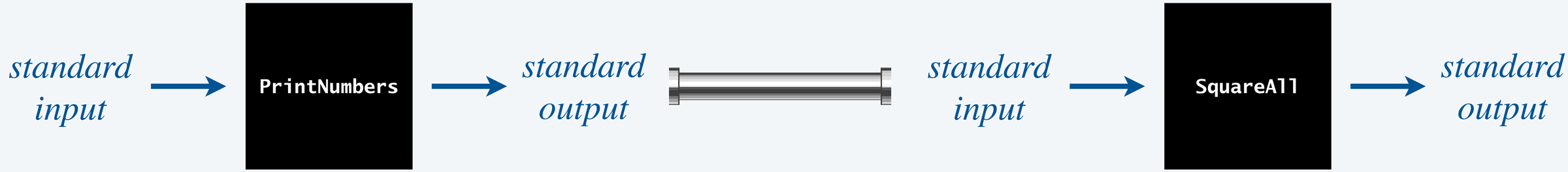
redirect standard output

filename

display content of a file

Piping

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



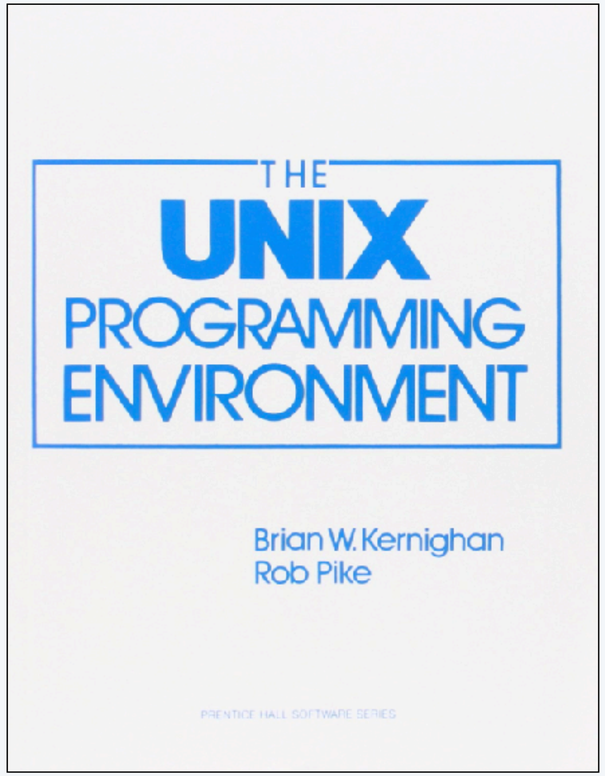

```
~/cos126/io> java-introcs PrintNumbers 3 | java-introcs SquareAll
1
4
9
~/cos126/io> java-introcs PrintNumbers 100 | java-introcs Average
50.5
```

pipe operator

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

Unix philosophy

- “Make every program do one thing (and that one thing) well”
- Unix-based programs – cat, ls, grep, awk, sed, sort, uniq, etc.
- Use piping to create quick data flows




```
kevinnegy@COS-MGXYTF9C72:~/Downloads/precept9$ head -n 1 healthcare_data.txt
Id AGE RACE ETHNICITY GENDER ADDRESS ZIP HEALTHCARE_EXPENSES HEALTHCARE_COVERAGE
kevinnegy@COS-MGXYTF9C72:~/Downloads/precept9$ cat healthcare_data.txt | sort -k 2 | awk '$2==2{print}' | wc -l
       3
```

Input and output: quiz 4

What is the output of the following command?

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

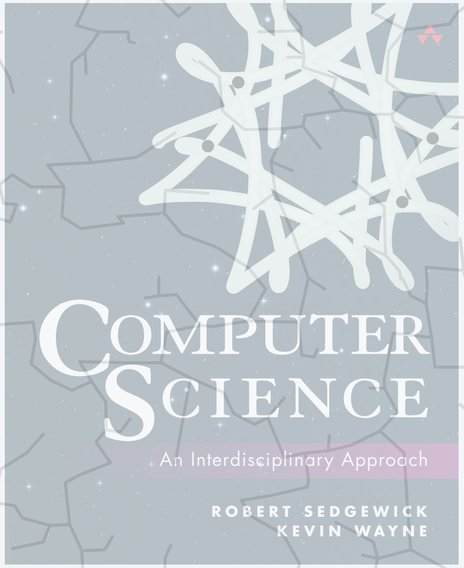
A. Integers from 1 to 100.

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*

Standard MIDI library

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

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*
static void	pause()	*pauses for the specified duration*
static void	noteOff()	*turns specified note off*
⋮	⋮	⋮

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!*

Recap: What are some limitations of our existing text IO methods?

- Non-interactive
  - StdIn
- Technically limited list of command line arguments
  - StdIn
- What if we want to read input from file?
  - Redirection
- What if we want to store output for later?
  - Redirection
- What if we want to send output to another application?
  - Pipes

Credits

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