CSE 143
Lecture 9

Recursion

slides created by Alyssa Harding
http://www.cs.washington.edu/143/

Recursion

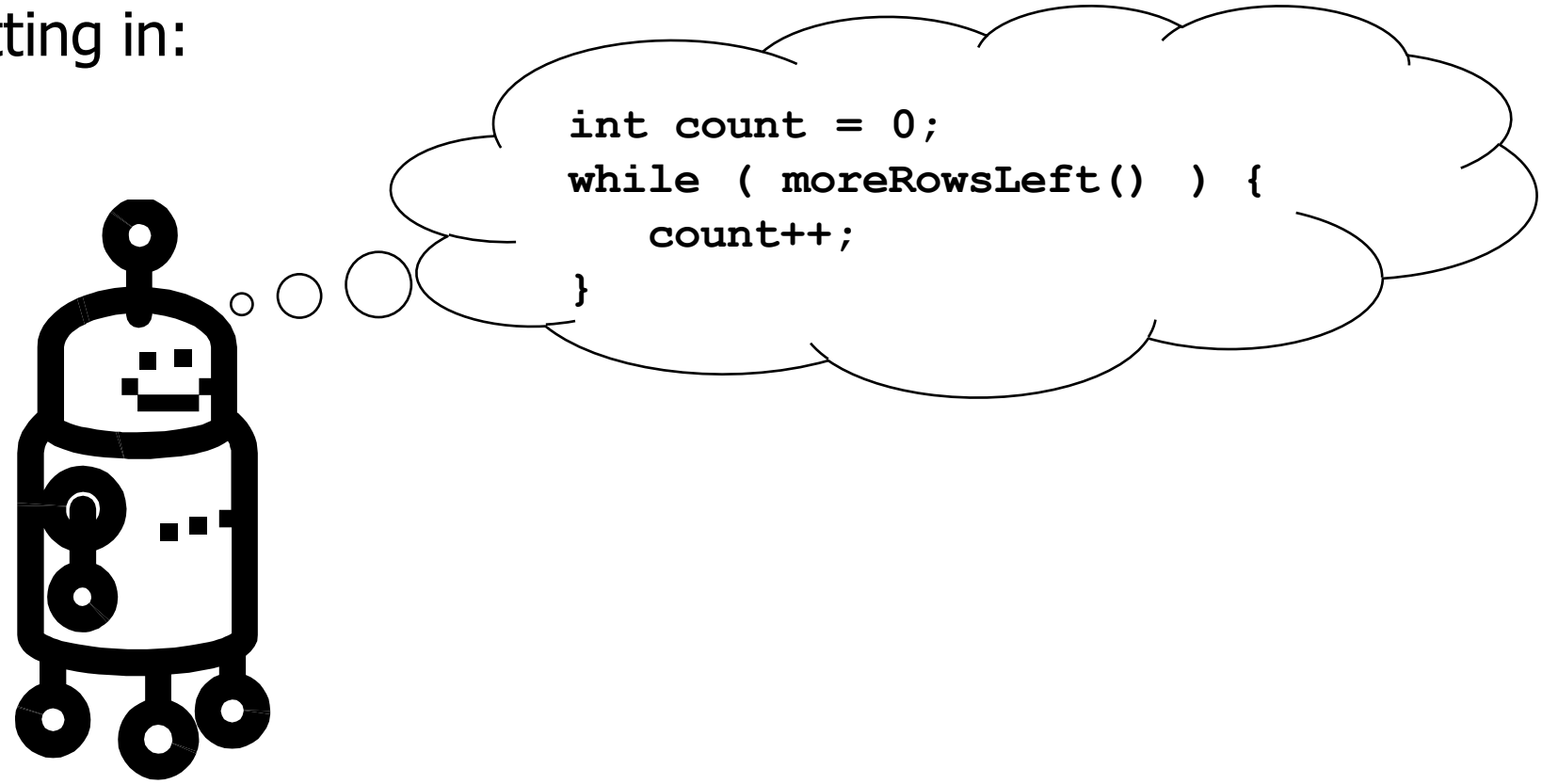
- **Iteration**: a programming technique in which you describe actions to be repeated using a loop

- **Recursion**: a programming technique in which you describe actions to be repeated using a method that calls itself

- Both approaches can be used to solve many of the same problems
  - Some problems are easier solved **iteratively**
  - Some problems are easier solved **recursively**

Example: row

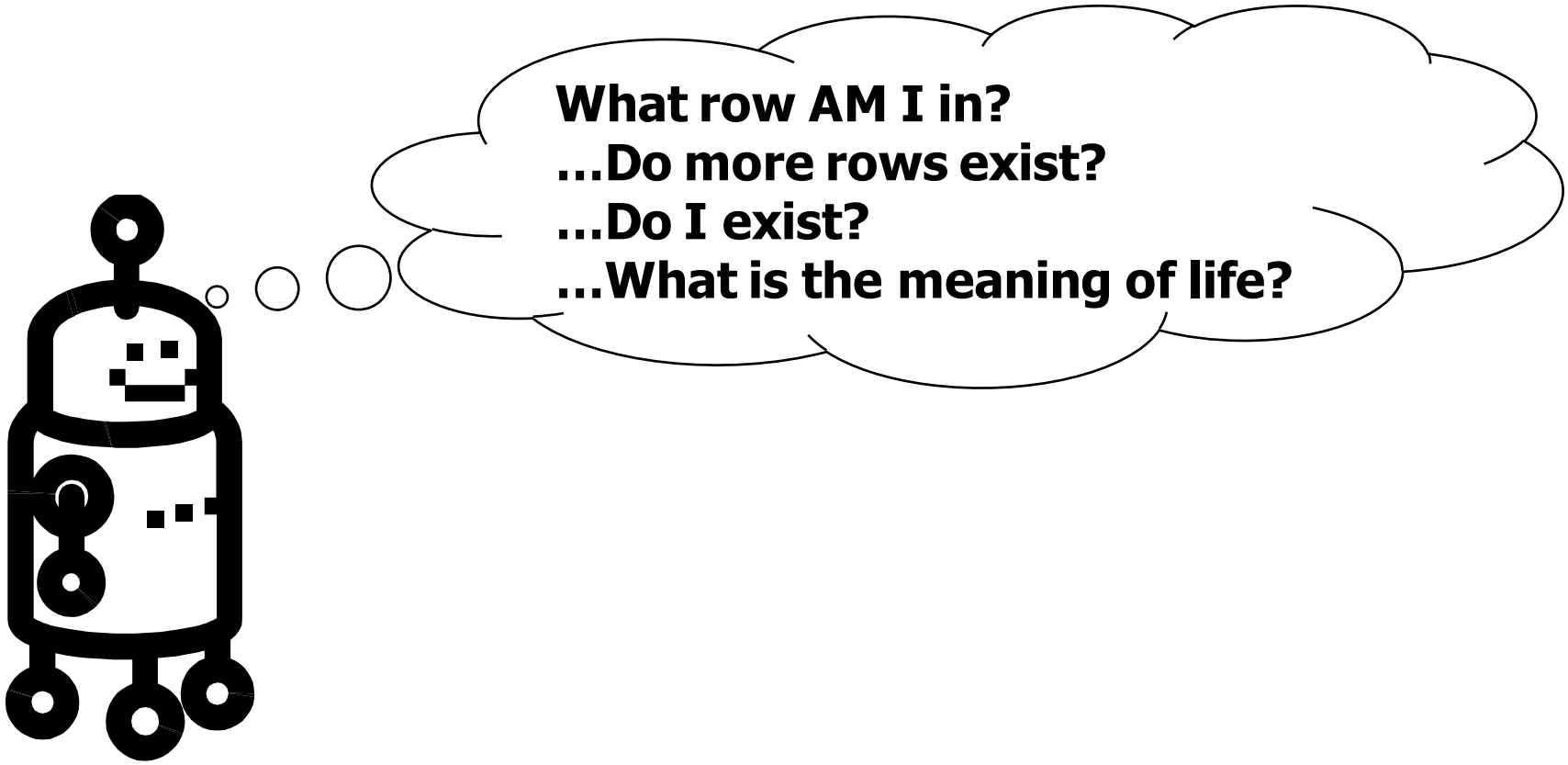
- Imagine that you're a robot and I ask you which row you're sitting in:



- So far, you're programmed to take an iterative approach

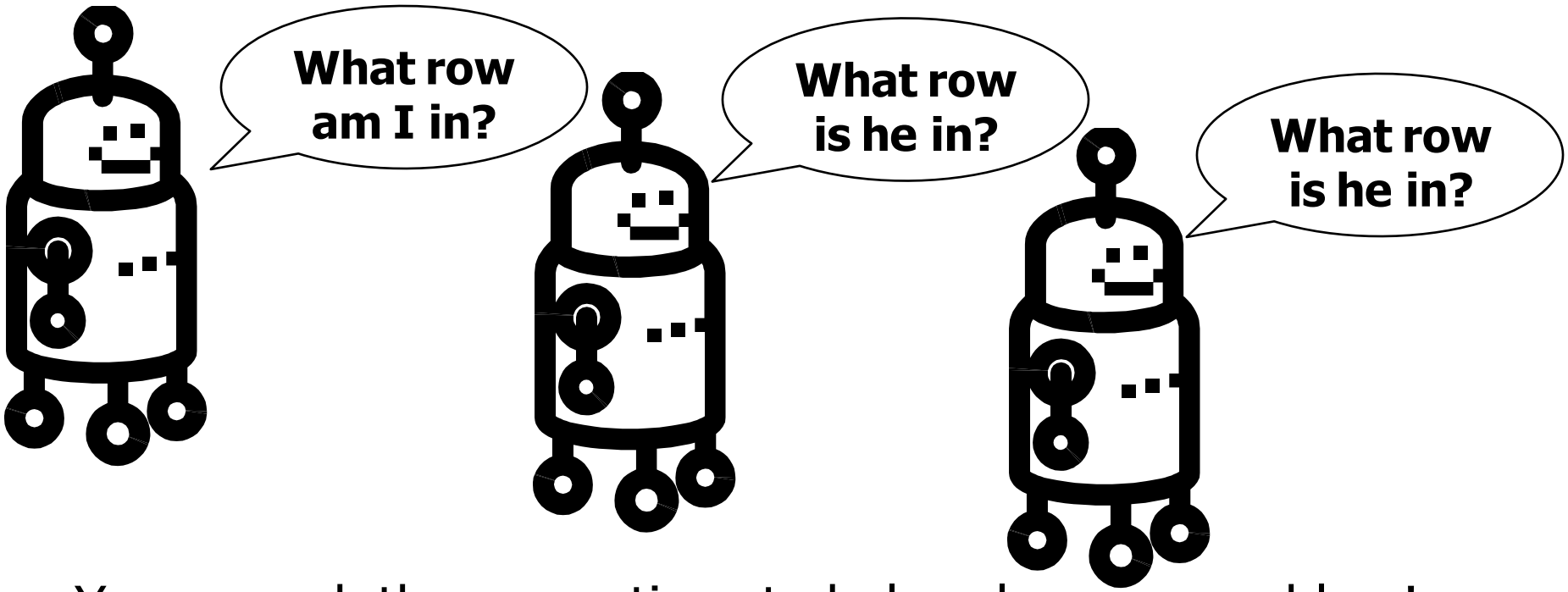
Example: row

- What if you're a robot who can't see well?



Example: row

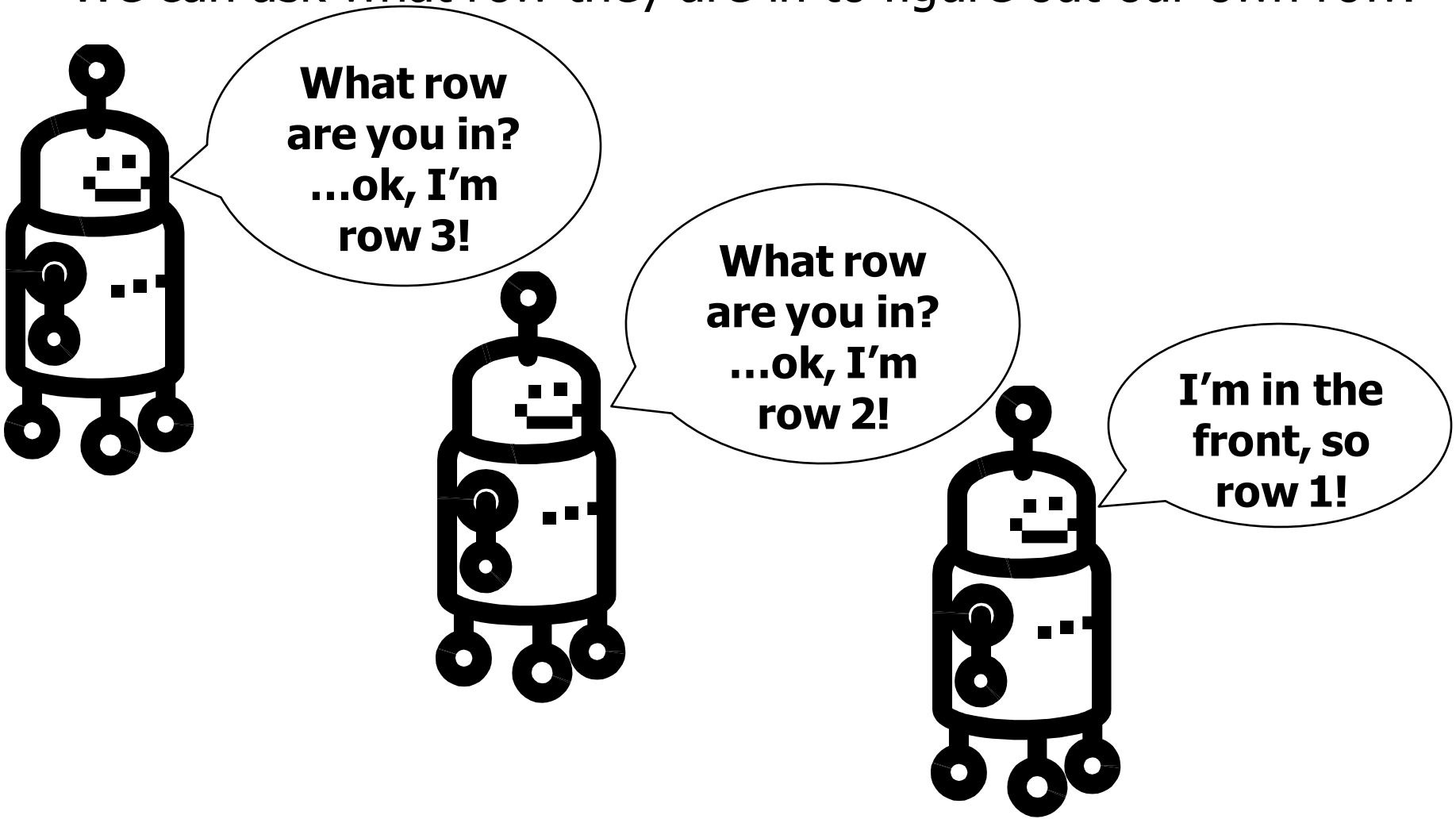
- What if you're have a room full of other robots?



- You can ask them questions to help solve your problem!
- …but not that question! We need to make progress each time

Example: row

- We can ask what row they are in to figure out our own row:



Case analysis

- Iteratively, we think of the loop bounds
- Recursively, we think of the cases
- Base case:
  - Easiest, simplest case where we know exactly what work to do
  - Example: "If I'm in the front row, I'm in row 1."
- Recursive case:
  - We do a little bit of work and ask someone else a simpler version of the same question
  - Example: "Otherwise, I ask the person in front of me what row they are in and add 1!"

Case analysis

- Key questions to ask:

- Identifying the base case:
  - What is the easiest case?
  - When do I know that I'm done?

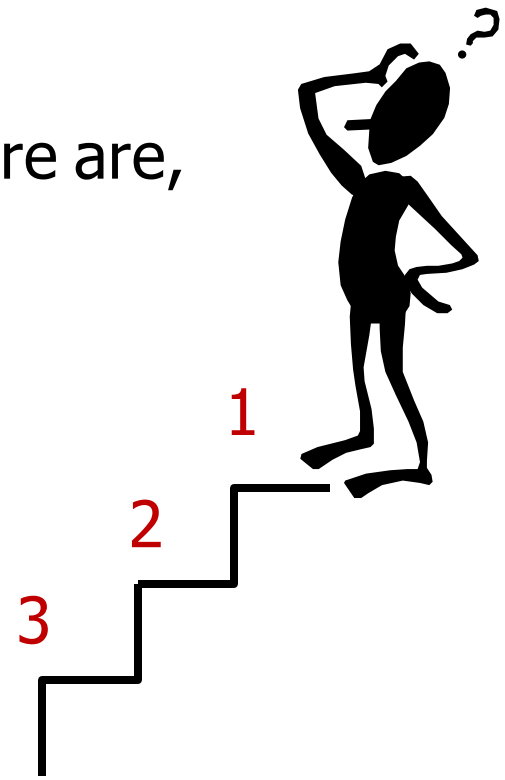
- Working out the recursive case:
  - What's a small bit of work that I can do?
  - What progress can I make towards my goal?
  - Is there a repeated pattern?

Example: stairs

- You want to walk down a flight of stairs.

- Iterative approach:

"Let me count the number of stairs there are, and then take that that many steps!"



Example: stairs

- You want to walk down a flight of stairs.

- Recursive approach:

"If I'm at the bottom, I stop.
Otherwise, I take a step down and repeat."

Example: `writeStars`

- Here's an iterative approach to making a method that writes out `n` stars:

```
public static void writeStars(int n) {
   for (int i = 0; i < n; i++)
     System.out.print("*");
   System.out.println();
 }
```

Example: `writeStars`

- Let's transform it to be recursive!

- What is the base case?

```
public static void writeStars2(int n) {
    if ( n == 1 ) {
        System.out.println("*");
    } else {
        ...
    }
}
```

Printing 1 star is easy,
but printing 0 is even easier!

Example: `writeStars`

- Let's transform it to be recursive!

- What is the base case?

```
public static void writeStars2(int n) {
    if ( n == 0 ) {
       System.out.println();
    } else {
       ...
    }
}
```

Here's our simplest base case.

Example: `writeStars`

- Let's transform it to be recursive!

- What is the recursive case?

```
public static void writeStars2(int n) {
    if ( n == 0 ) {
        System.out.println();
    } else {
        for (int i = 0; i < n; i++ ) {
            System.out.println("*");
        }
    }
}
```

We're a lazy robot! We just want to make a small amount of progress.

Example: `writeStars`

- Let's transform it to be recursive!

- What is the recursive case?

```
public static void writeStars2(int n) {
    if ( n == 0 ) {
        System.out.println();
    } else {
        System.out.println("*");
        writeStars2(n - 1);
    }
}
```

We make a little progress...

We ask another robot to do the rest.

We have to trust that we're writing the method well!

Example: `writeStars`

- We can trace its progress as it goes:

```
writeStars2(3)
    System.out.print("*")
    writeStars2(2)
        System.out.print("*")
        writeStars2(1)
            System.out.print("*")
            writeStars2(0)
                System.out.println()
```

Example: `reverse`

- Now we'll look at a problem that's hard to solve iteratively, but easier with recursion

- Given a **`Scanner`** as input, print the lines in reverse

- How would you solve this iteratively?
  - Loop while there are more lines
  - Requires additional storage, like a **`List`** or a **`Stack`**

Example: `reverse`

- Writing **`reverse`** recursively:

- What is the base case?

```
public static void reverse(Scanner input) {
    // base case: no more lines
    if ( !input.hasNextLine() ) {
        // do nothing
    } else {
        …
    }
}
```

This is a good base case,
but we don't need to
do anything in this case

Example: `reverse`

- Writing **`reverse`** recursively:

- What is the base case?

```
public static void reverse(Scanner input) {
    // base case: no more lines
    // recursive case
    if ( input.hasNextLine() ) {
        …
    }
}
```

It's better style not to have an empty if statement.

Example: `reverse`

- Writing **`reverse`** recursively:

- What is the recursive case's work?

```
public static void reverse(Scanner input) {
    // base case: no more lines
    // recursive case
    if ( input.hasNextLine() ) {
        String line = input.nextLine();
        // reverse the rest of the input
        System.out.println(line);
    }
}
```

We made a little progress, how do we do the rest?

Example: `reverse`

- Writing **`reverse`** recursively:

- What is the recursive case's work?

```
public static void reverse(Scanner input) {
    // base case: no more lines
    // recursive case
    if ( input.hasNextLine() ) {
        String line = input.nextLine();
        reverse(input);
        System.out.println(line);
    }
}
```

We recursively call the method with the easier problem!

Example: `reverse`

```
public static void main (String[] args) {
public static void reverse(Scanner input) {
public static void reverse(Scanner input) {
public static void reverse(Scanner input) {
public static void reverse(Scanner input) {
public static void reverse(Scanner input) {
    if ( input.hasNextLine() ) {  // false!
        String line = input.nextLine();
        reverse(input);
        System.out.println(line);
    }
}
}
}
}
}
```

Input:

```
student
that
loves
recursion
```

```
recursion
loves
that
student
```

Example: `stutter`

- Our favorite problem: stutter!

- Given an **`int`** as input, stutter the digits
  - Example: **`stutter(348)`** returns **`334488`**

- So far we've only printed inside of our recursive methods, but we can return values as well

Example: `stutter`

- What is the base case?

```
public static int stutter(int n) {
  if ( n < 10 ) {
    return n*11;
  } else {
    ...
  }
}
```

Any single digit number can be stuttered easily.

Example: `stutter`

- What is the recursive case?

```
public static int stutter(int n) {
  if ( n < 10 ) {
    return n*11;
  } else {
    ...
  }
}
```

We can make a smaller problem by breaking the number down:

```
n = 348        n/10 -> 34
               n%10 ->  8
```

and recurse by stuttering both parts:

```
stutter(n/10) -> 3344
stutter(n%10) ->   88
```

Example: `stutter`

- What is the recursive case?

```
public static int stutter(int n) {
  if ( n < 10 ) {
    return n*11;
  } else {
    return stutter(n/10)*100 + stutter(n%10);
  }
}
```

To put them back into one number, we can't just add. We need to shift the first digits to the right:

```
stutter(n/10)*100 + stutter(n%10)
      3344*100 +              88
```

Example: `stutter`

- What about negative numbers?

```
public static int stutter(int n) {
  if ( n < 0 ) {
    return -stutter(-n);
  } else if ( n < 10 ) {
    return n*11;
  } else {
    return stutter(n/10)*100 + stutter(n%10);
  }
}
```

We deal with them first and trust the recursion to take care of the rest.