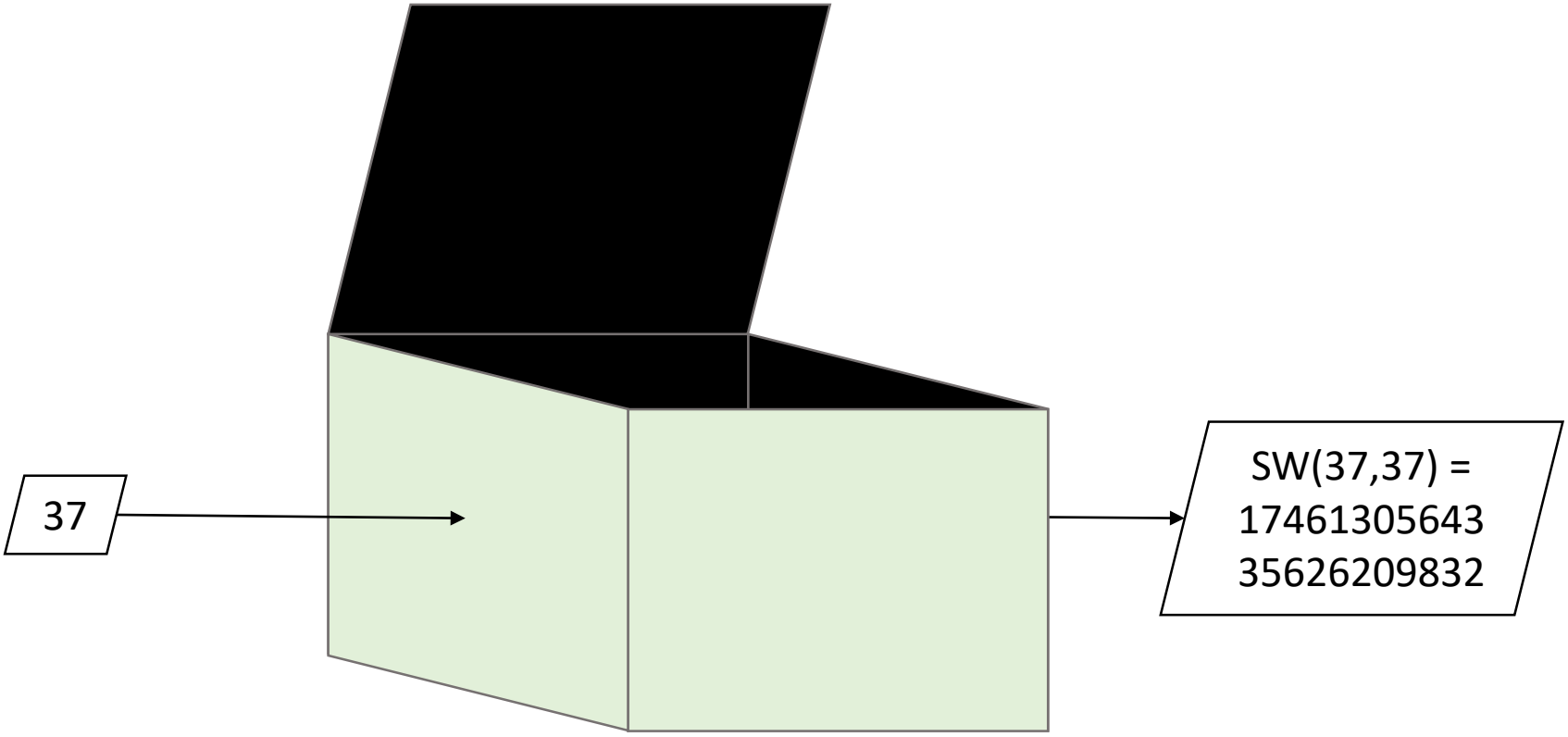
# What CAN’T Algorithms Do?



# What is an algorithm?

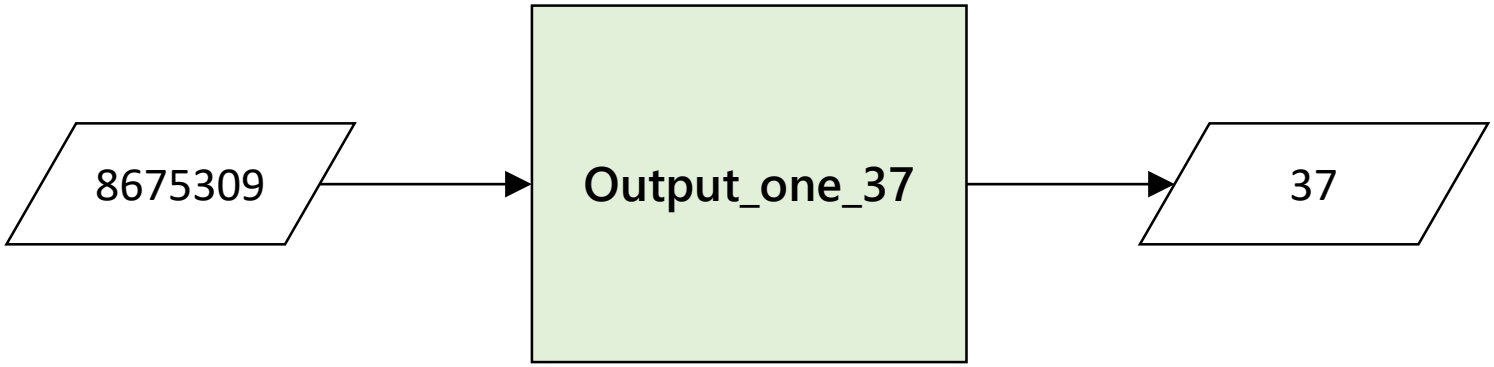
- Definition from Week 1:
  - a sequence of unambiguous instructions
  - for obtaining a required output
  - for any legitimate input
  - in a finite amount of time

# A program to output “37”

- Input: any number
- Output: 37

```
public class Output_one_37 {
    public static void main(String[] args) {
        System.out.println(37);
    }
}
```

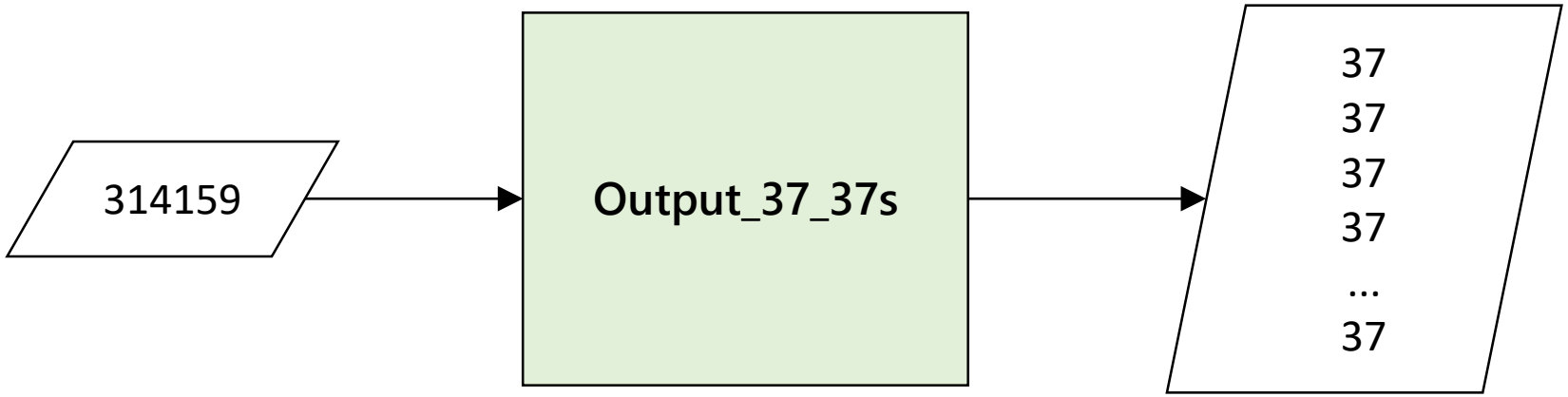
# Previous program as a black box*




* Green box.

# Another program – output more 37s

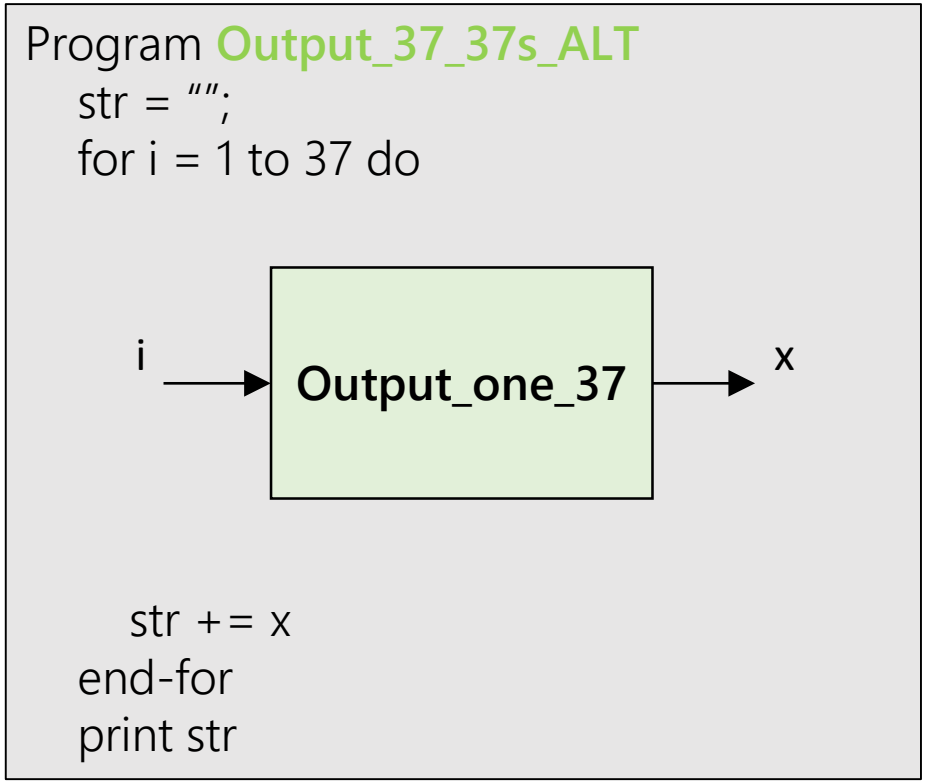
- Input: any number
- Output: thirty-seven 37s



# It might look like this:

```
public class Output_37_37s {
    public static void main(String[] args) {
        String str = "";
        for (int i = 1; i <= 37; i++) {
            str += "37\n";
        }
        System.out.println(str);
    }
}
```

# OR … it might look like THIS:




The point: programs can also be subroutines

# Recall: Optimization vs. Decision

Given a set of available items each with a weight and a value, and a knapsack with a carrying capacity of 37 pounds:

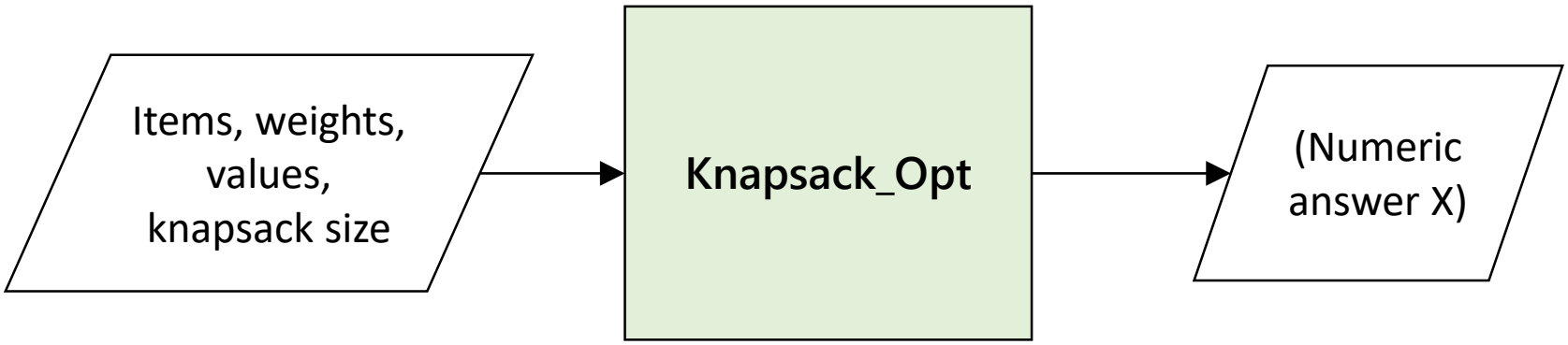
- What is the *maximum* total value of items that a thief can steal?

  Output: a number

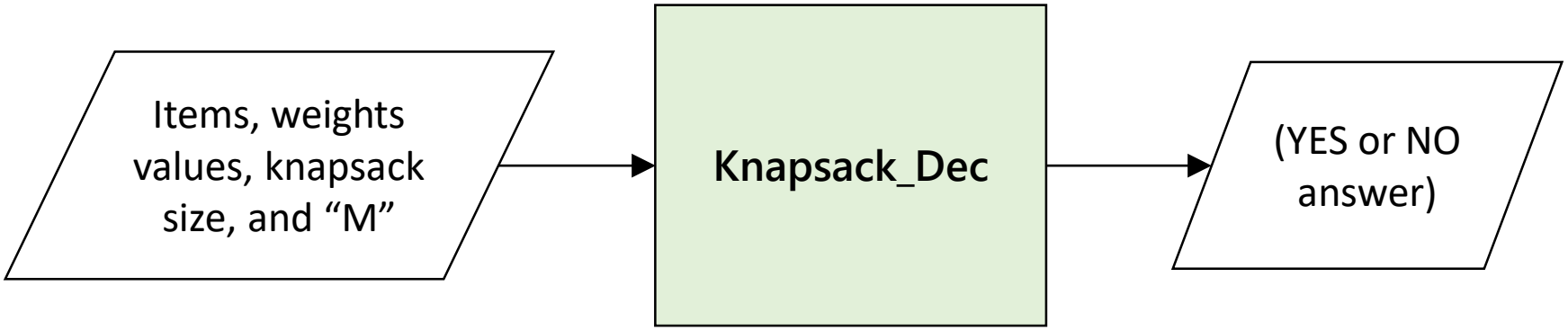
- Can the thief find a subset of items worth at least $42?

  Output: “YES” or “NO”

# A program for the optimization problem



# A program for the decision problem



Background/side info

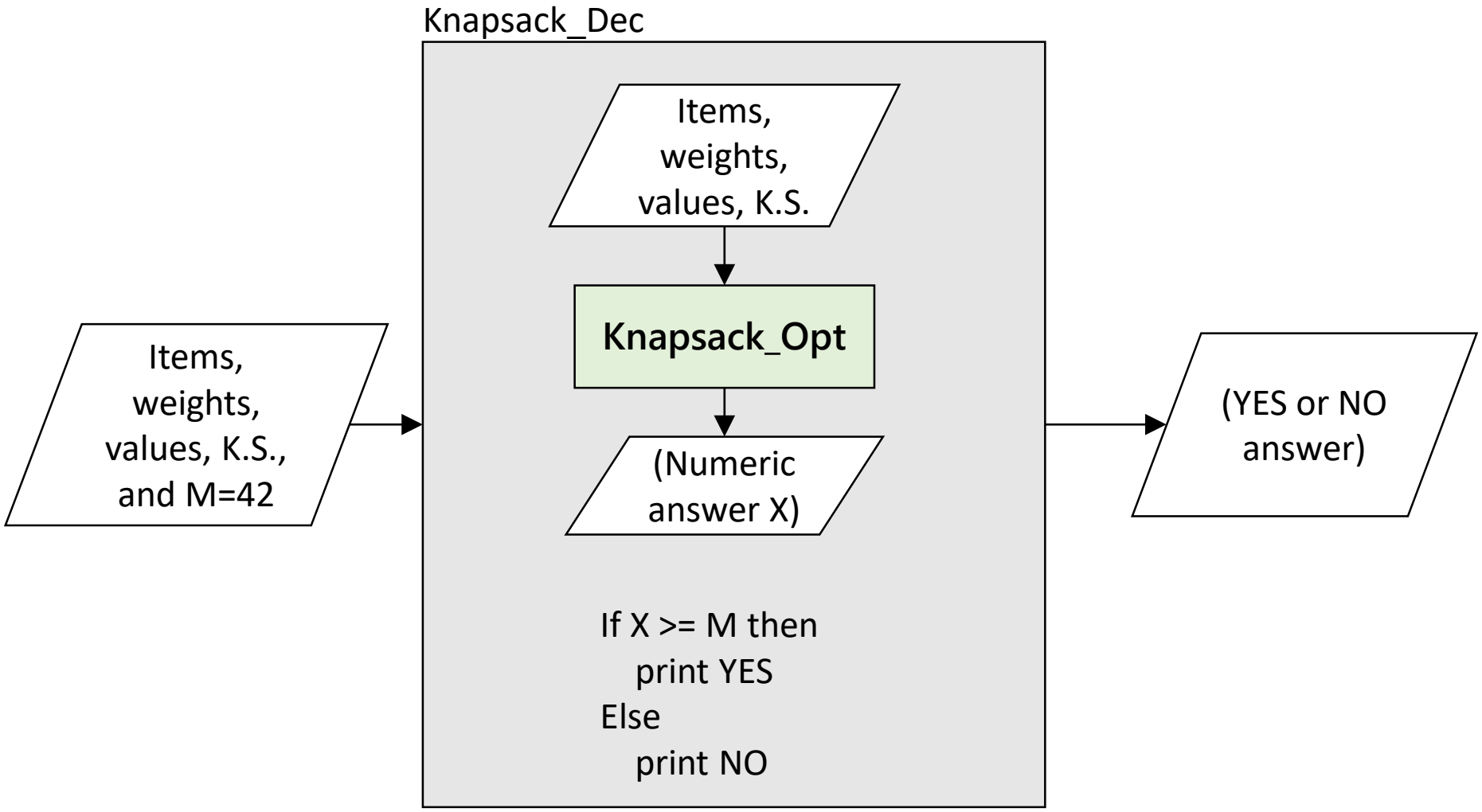
# So here's A Thing:

- If you know how to solve *either* the decision or the optimization problem, then you can solve the other one

- *"Knowing how"* to solve some problem means that you can use it as a subroutine (black box)

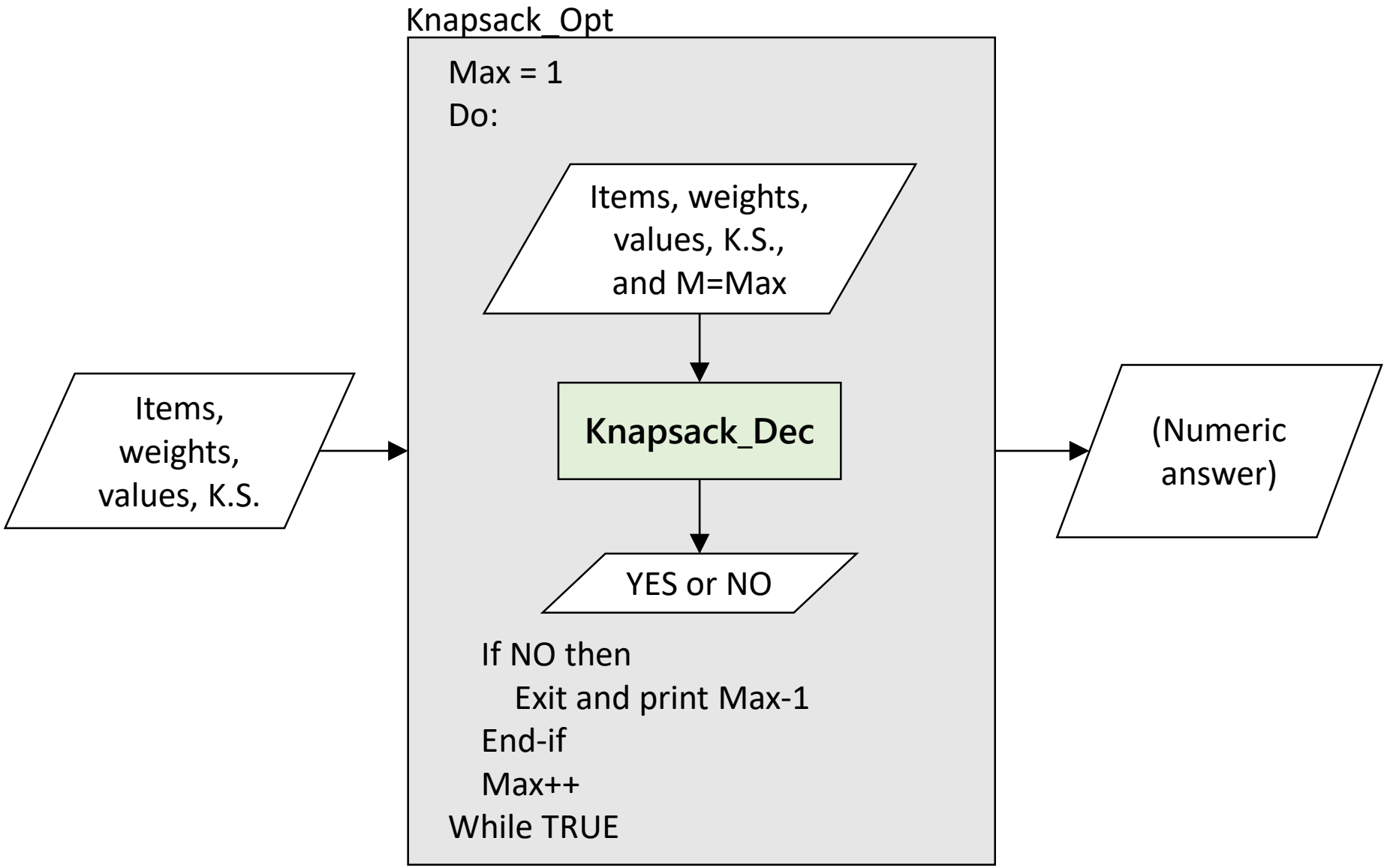
Background/side info

# Using Knapsack_Opt to solve Knapsack_Dec



Background/side info

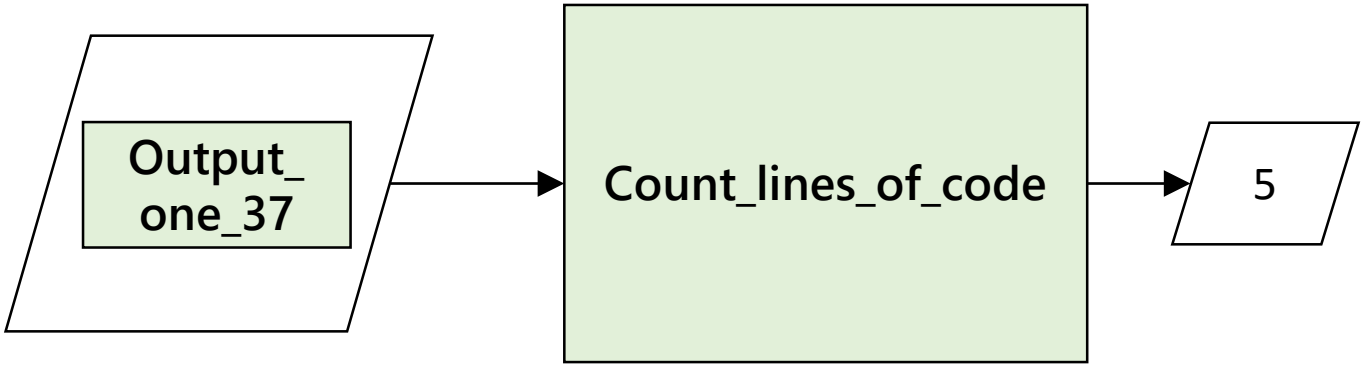
# Using Knapsack_Dec to solve Knapsack_Opt



Background/side info

# Here's another Thing:

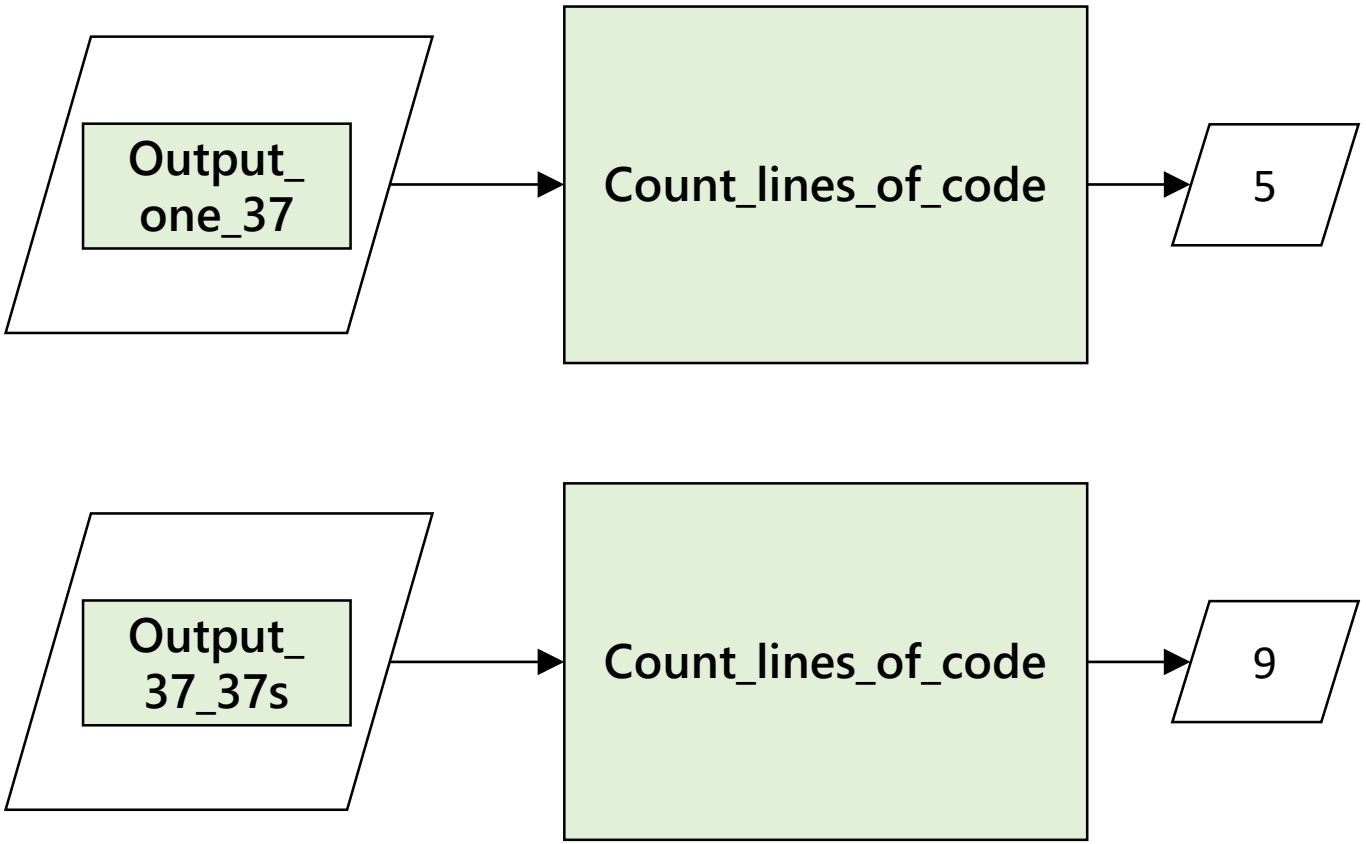
- Programs can also be "data"
- They are only character strings, after all



Background/side info

# Here's another Thing:

- Programs can also be "data"
- They are only character strings, after all



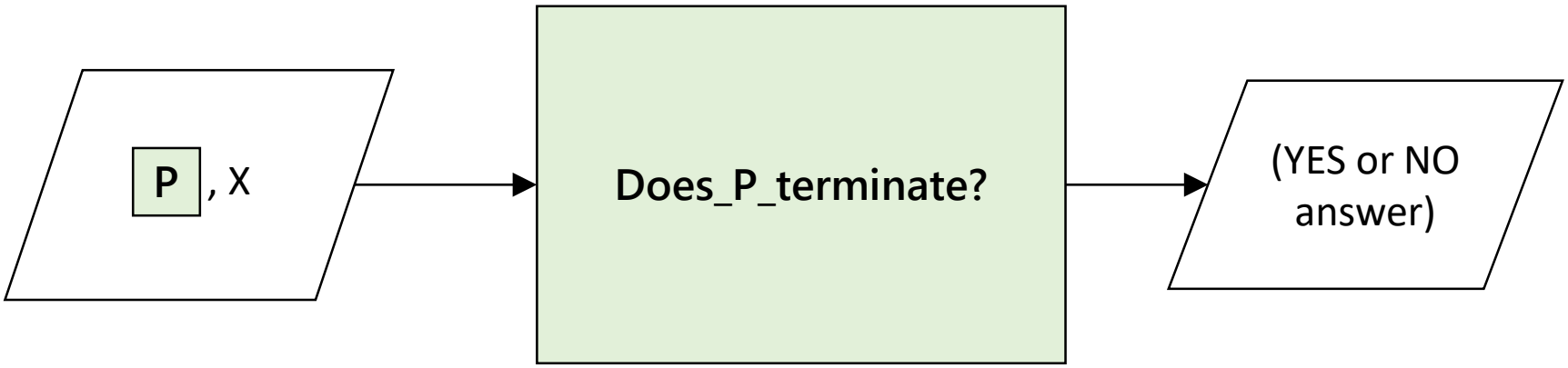
# Some programs never terminate

- This is not a good thing

```
public class Uh_oh {
    public static void main(String[] args) {
        boolean x = true;
        while (x) {
            int y = 42;
        }
        return 37;
    }
}
```

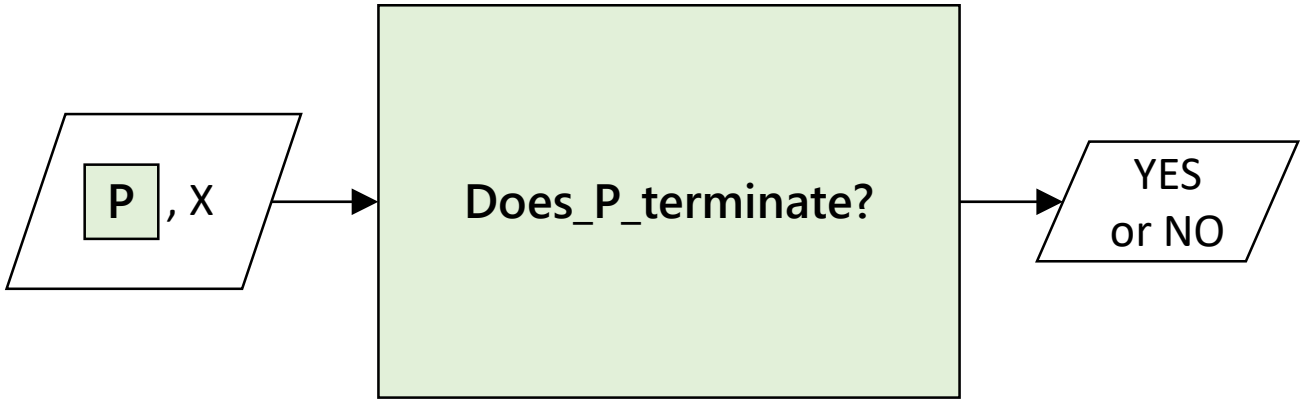
- Not in COMP labs, and not in real life

# This black box would be VERY useful:




- It's just a decision problem:
  - Given a program P and an input x, does P ever terminate?
- This problem is known as **The Halting Problem**
- Can we write an algorithm for it?

# The bad news




- This program *CANNOT* exist ☹
- We will prove this by contradiction

# Wait, why doesn't this work?

```
Program Does_P_terminate

    Step 1:
        Run P(x) (Emulator, Sandbox, VM, etc.)
    Step 2:
        If P terminated then
            Print YES
        else
            Print NO
        end-if-else
```

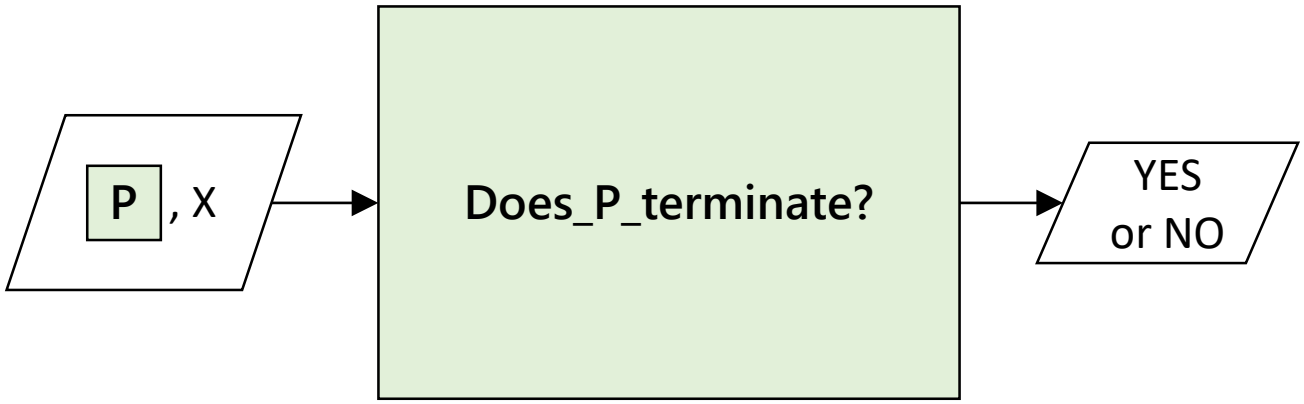
- An algorithm must finish in finite time (for any input)

# Proof by contradiction

- Assume that the thing you want to prove is false
- Show that this assumption leads to a logical contradiction
- Therefore, the thing must be true

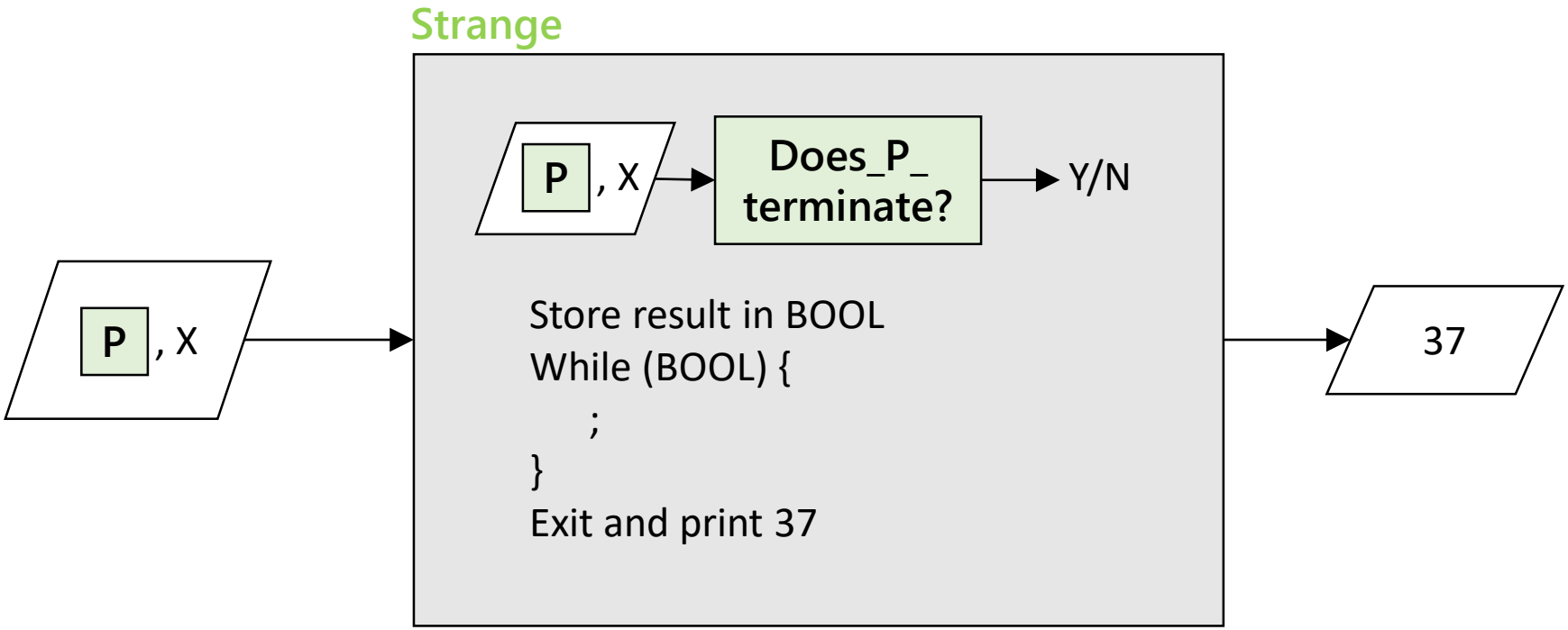
# Proof step 1

- Assume this program exists:



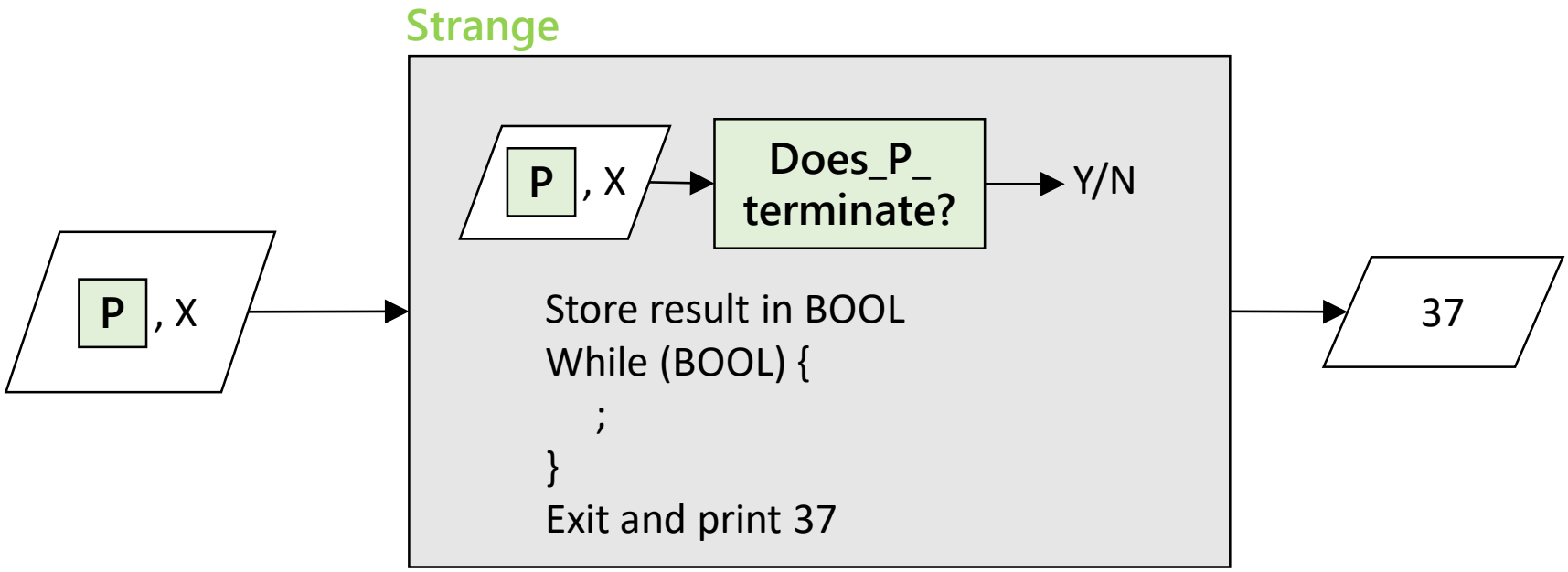
# Proof step 2

- Then we can write this strange new program:




The point here, really, is whether or not “Strange” will terminate (exit)

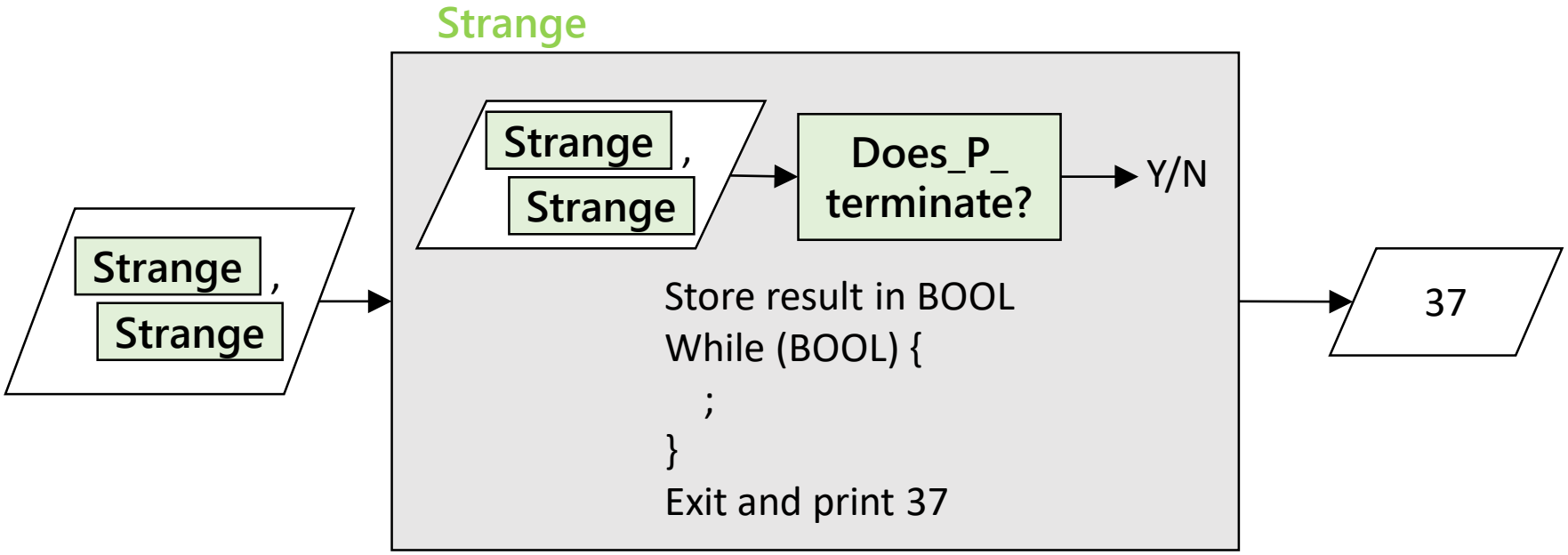
# This is Strange



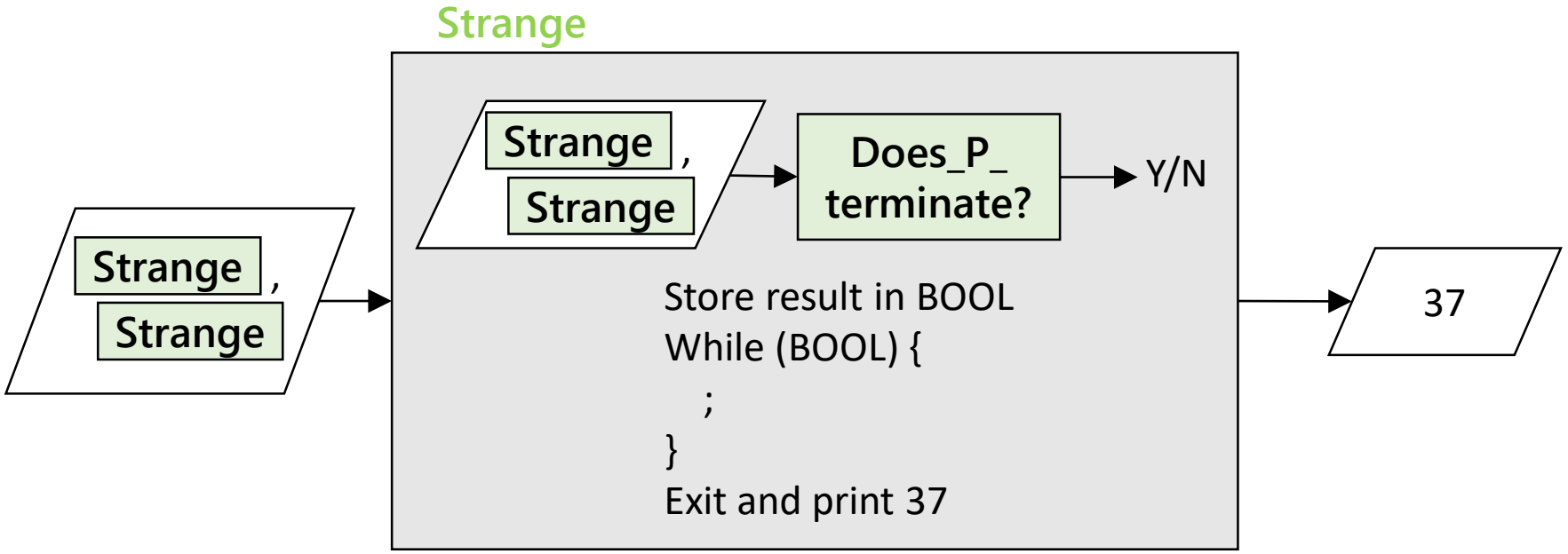

- If P terminates on input X, then Strange *does not* terminate
- If P *does not* terminate on input X, then Strange terminates

# Proof step 3

- Since “Strange” is just another program, we can feed it to itself!



# This is even stranger




- If Strange terminates, then Strange does not
- If Strange does not terminate, then Strange does
- This is impossible – a contradiction.

# Summary of the proof

- If we assume Does_P_terminate exists …
  - … then the program Strange also exists
- But the program Strange cannot exist
  - It would both terminate and *not* terminate at the same time
- Therefore Does_P_terminate does not exist

# The Halting Problem

- We have shown that this decision problem:
  - "Given a program P and input x, does P ever terminate?"
- … cannot be solved by any algorithm
- The Halting Problem is *undecidable*
- We are not the first to prove this
  - https://en.wikipedia.org/wiki/Alan_Turing
- The Page of Futility:
  - https://en.wikipedia.org/wiki/List_of_undecidable_problems