[301] Iteration

Tyler Caraza-Harter

Learning Objectives Today

Reason about loops

- Motivation: need for repetition
- Condition and body of loop
- “while” syntax
- loops inside loops

Chapter 7 of Think Python

Understand common use cases

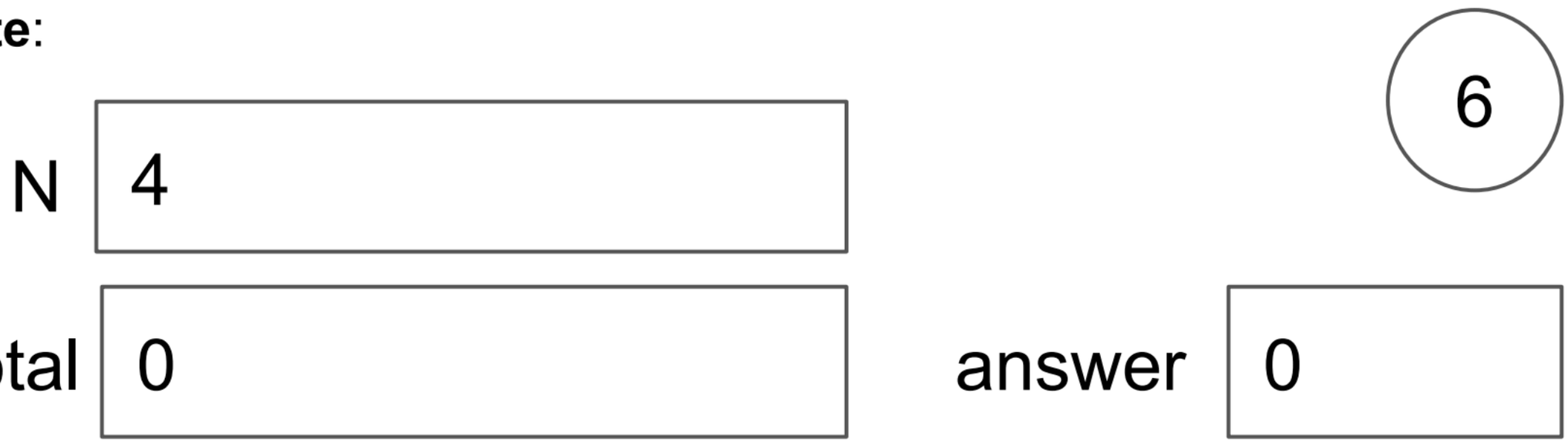
- Taking input from a user
- Computing over ranges of numbers

Learn to avoid pitfalls

- Infinite loops (when unintentional)
- Off-by-one mistakes

Worksheet

State:

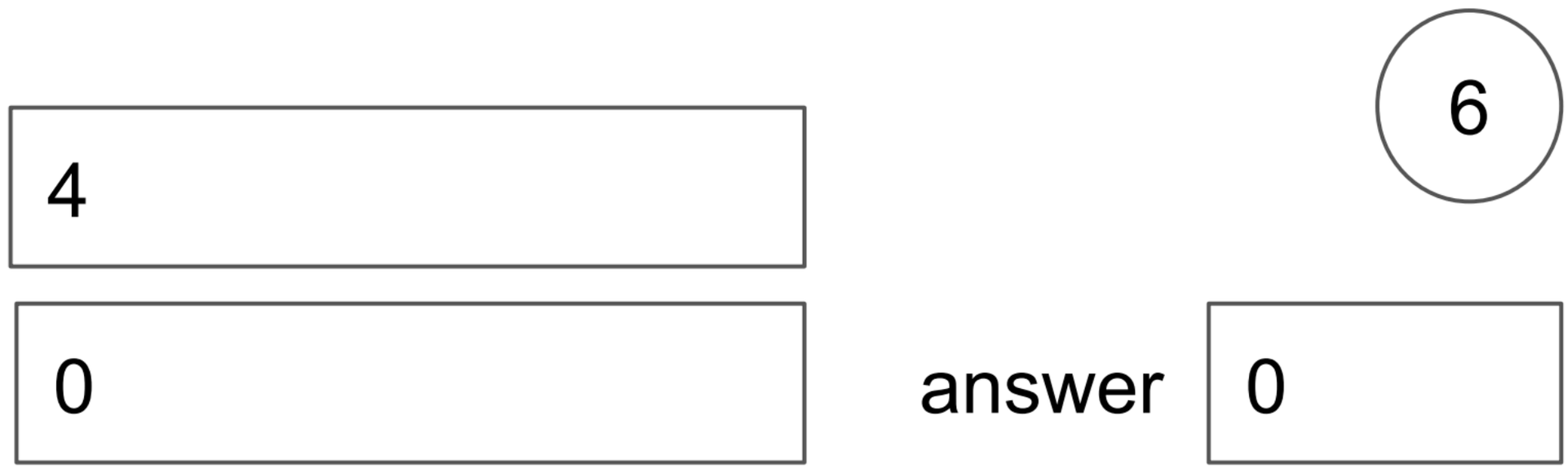


Code:

1. Put 1 in the “total” box
2. If “N” equals 1, skip to step 6, otherwise continue to step 3
3. Multiply the value in “total” by the value in “N”, and put the result back in “total”
4. Decrease the value in “N” by 1
5. Go to step 2
6. Copy the value in total to the answer box

Worksheet

State:



Code:

1. Put 1 in the “total” box
2. If “N” equals 1, skip to step 6, otherwise continue to step 3
3. Multiply the value in “total” by the value in “N”, and put the result back in “total”
4. Decrease the value in “N” by 1
5. Go to step 2
6. Copy the value in total to the answer box

Combination of conditionally skipping forward (2) with going back is (5) is called a “while loop”

Worksheet

State:

N | 4

total | 0

6

answer | 0

Code:

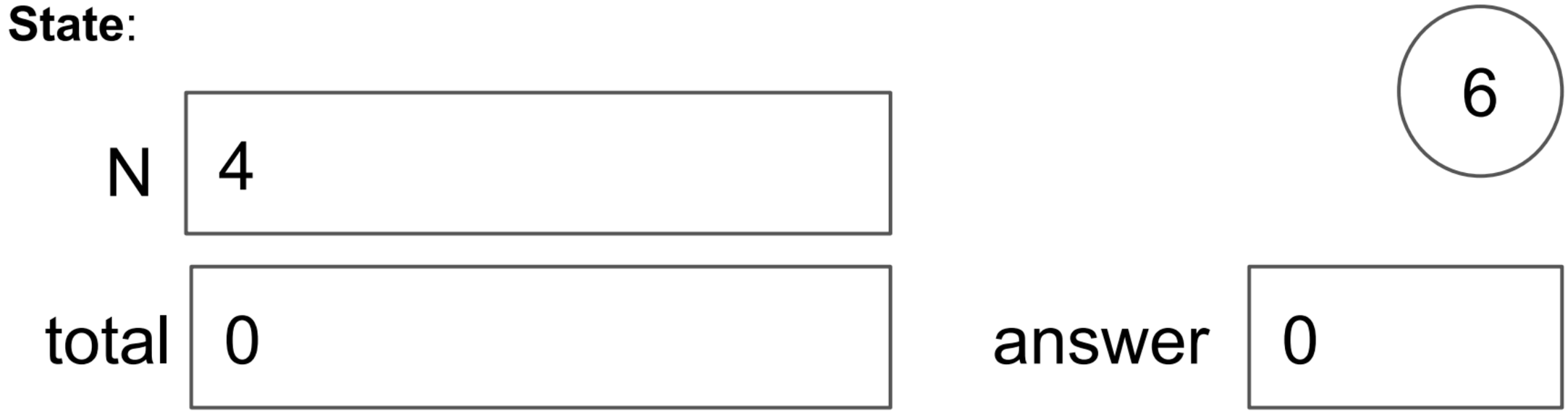
loop condition

1. Put 1 in the “total” box
2. If “N” equals 1, skip to step 6, otherwise continue to step 3
3. Multiply the value in “total” by the value in “N”, and put the result back in “total”
4. Decrease the value in “N” by 1
5. Go to step 2
6. Copy the value in total to the answer box

loop body

Worksheet

State:



Code:

loop condition

1. Put 1 in the “total” box
2. If “N” equals 1, skip to step 6, otherwise continue to step 3
3. Multiply the value in “total” by the value in “N”, and put the result back in “total”
4. Decrease the value in “N” by 1
5. Go to step 2
6. Copy the value in total to the answer box

loop body

going back will be implicit in Python, and will happen right after loop body.
you can identify the loop body because it will be indented

Worksheet

State:

N | 4

total | 0

answer | 0

6

Code:

1. Put 1 in the “total” box
2. If “N” equals 1, skip to step 6, otherwise continue to step 3
3. Multiply the value in “total” by the value in “N”, and put the result back in “total”
4. Decrease the value in “N” by 1
5. Go to step 2
6. Copy the value in total to the answer box

loop condition

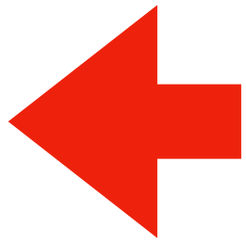
skip past loop body

continue to loop body

loop body

going back will be implicit in Python, and will happen right after loop body.
you can identify the loop body because it will be indented

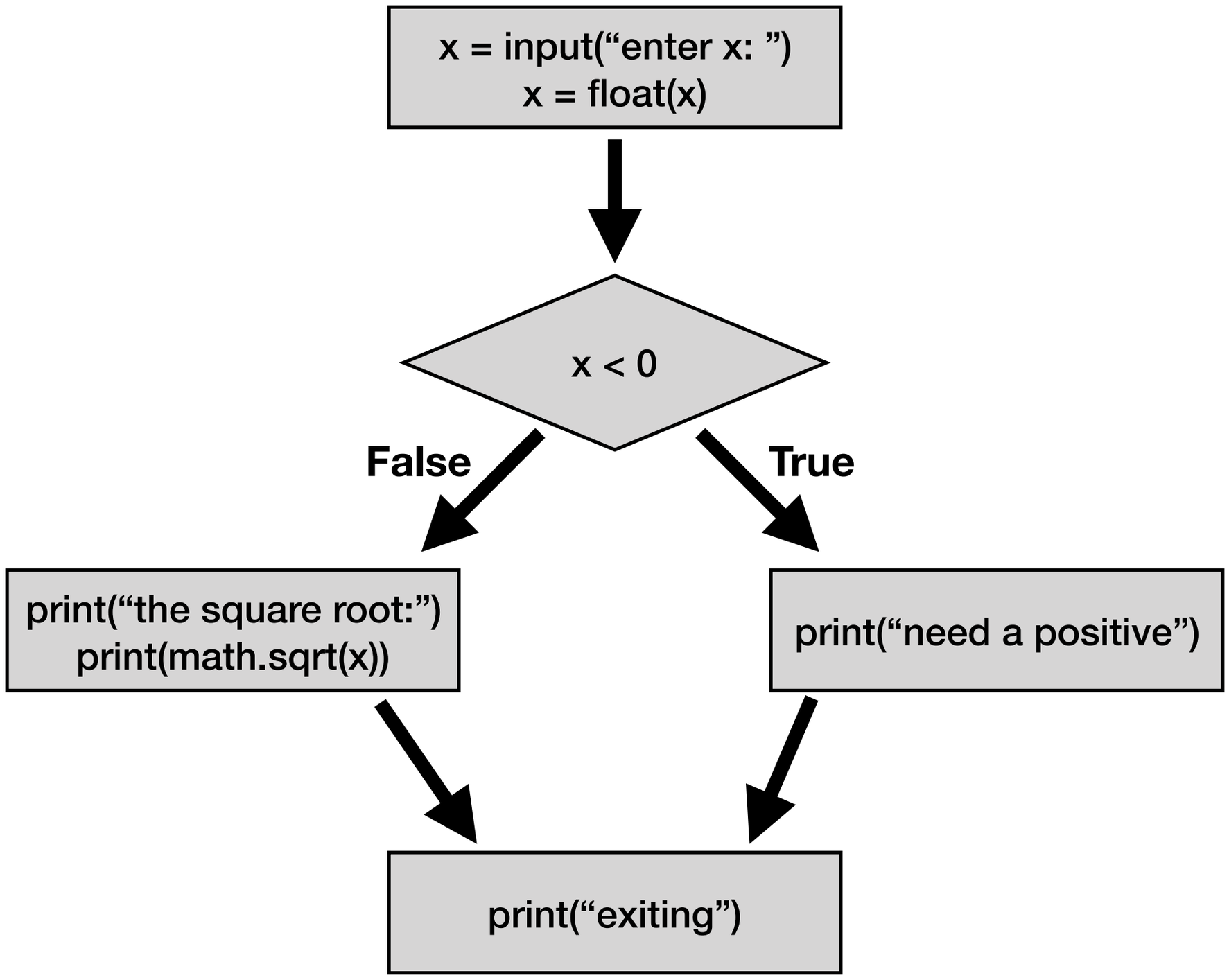
Today's Outline

Control Flow Diagrams 

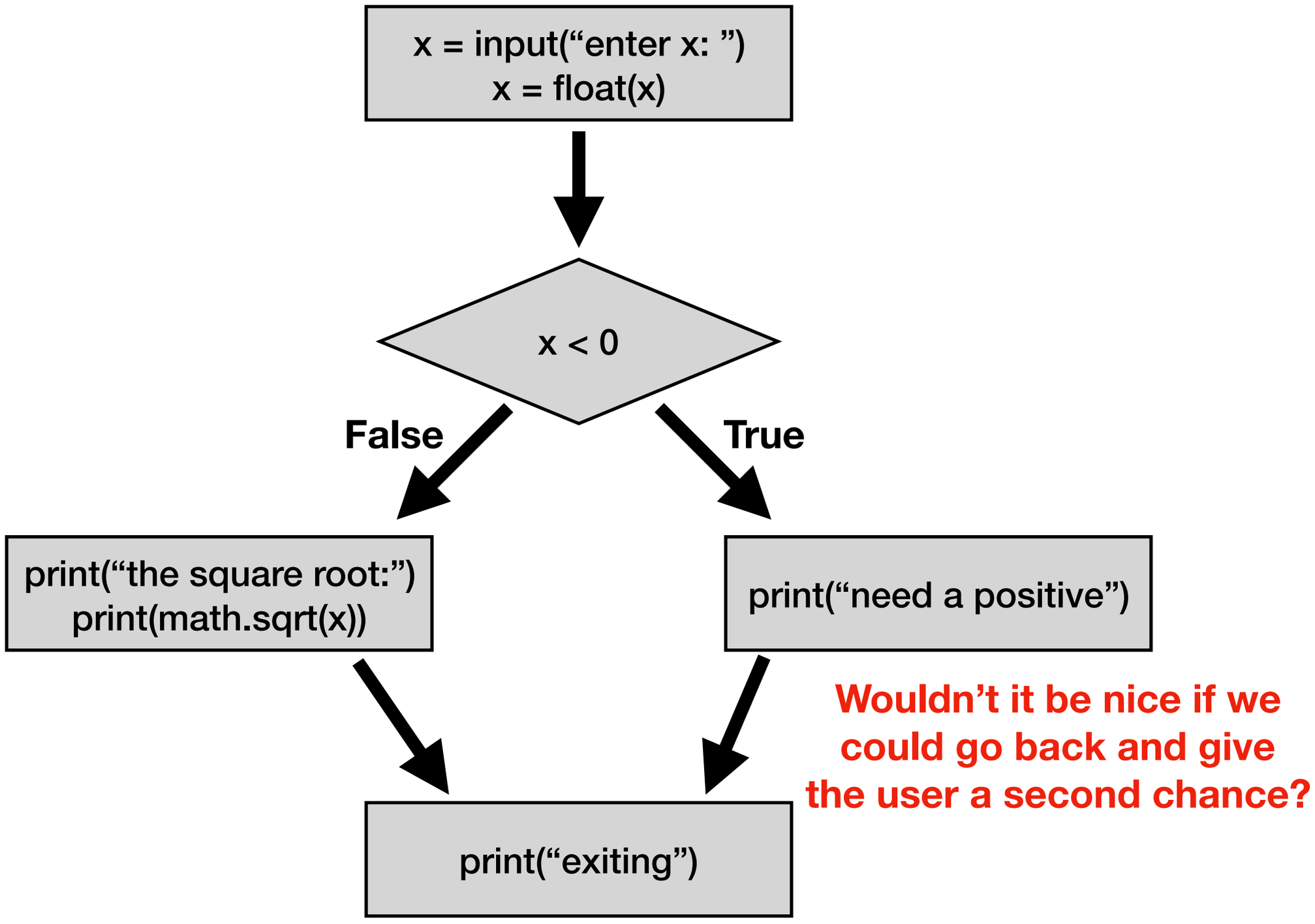
Basic syntax for “while”

Demos

Control Flow Diagrams: “if”

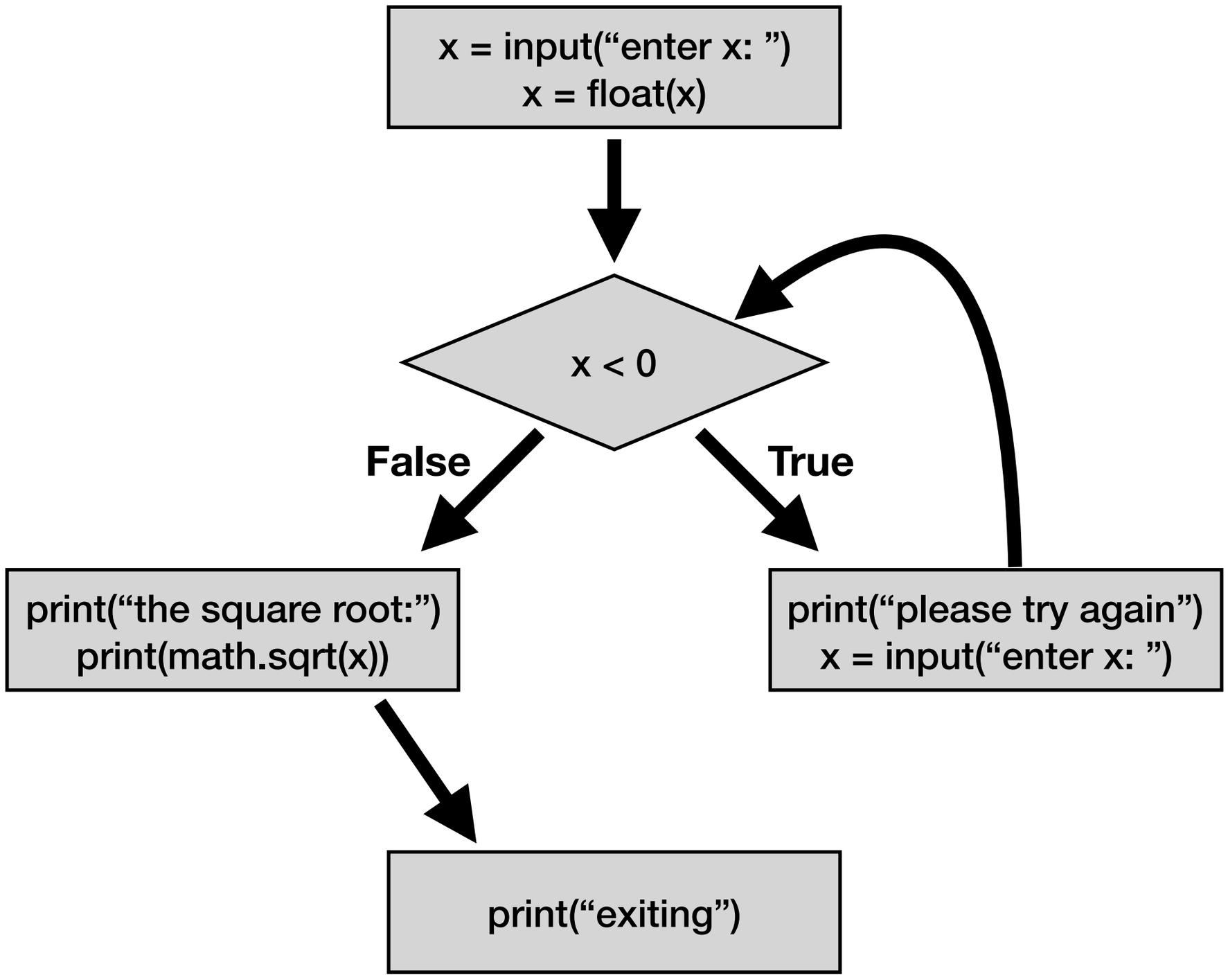


Control Flow Diagrams: “if”

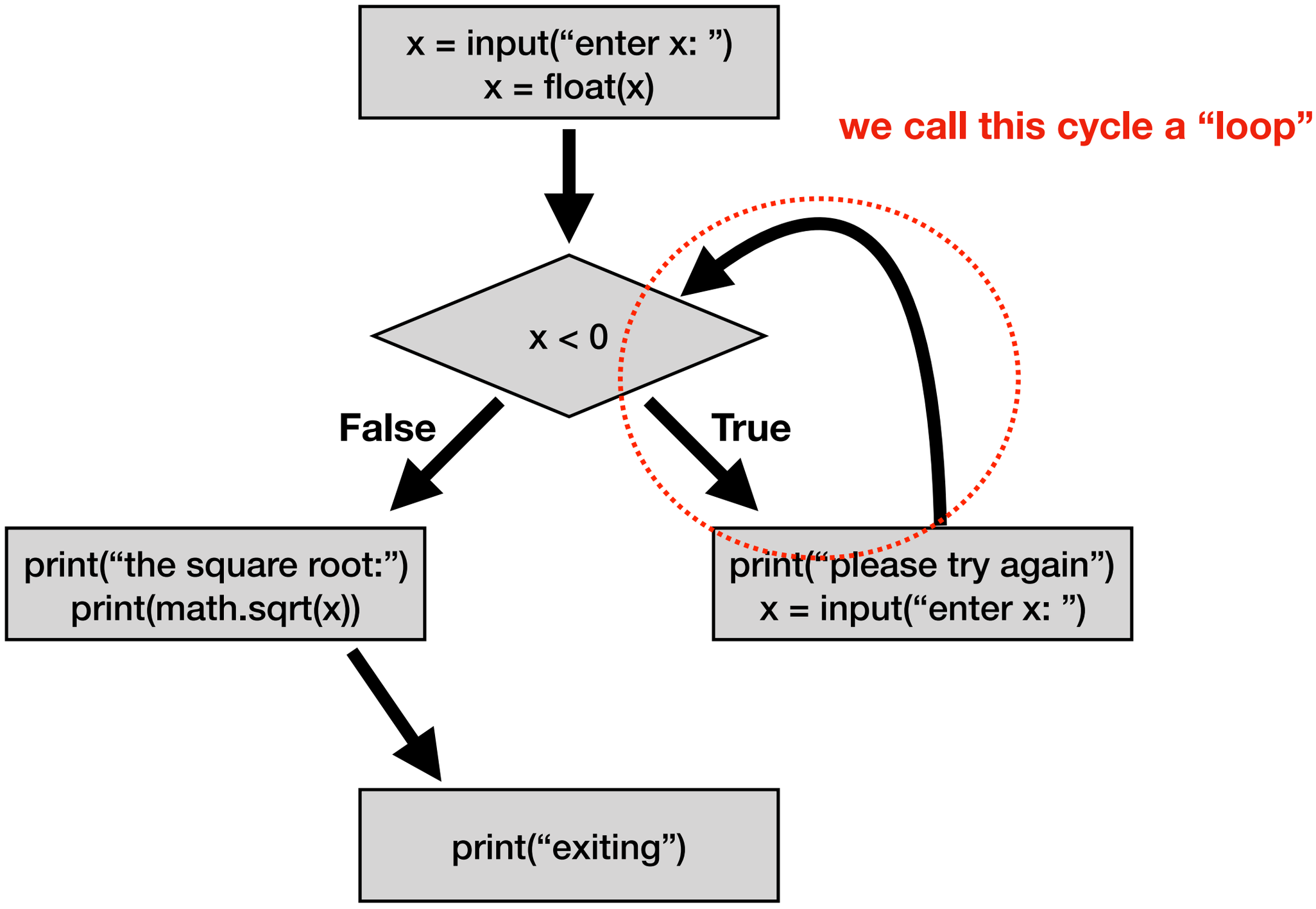


Wouldn’t it be nice if we could go back and give the user a second chance?

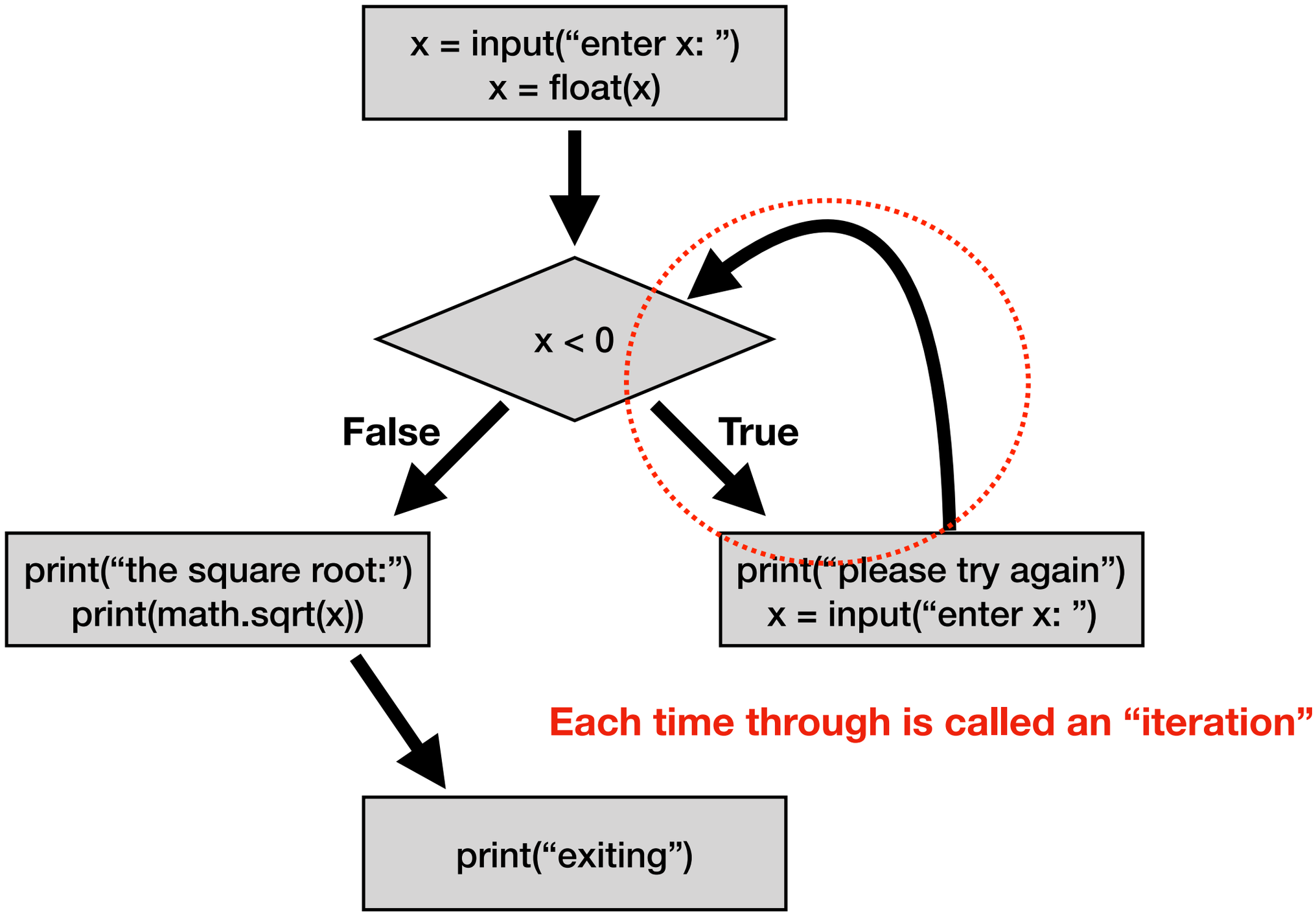
Control Flow Diagrams: “while”



Control Flow Diagrams: “while”

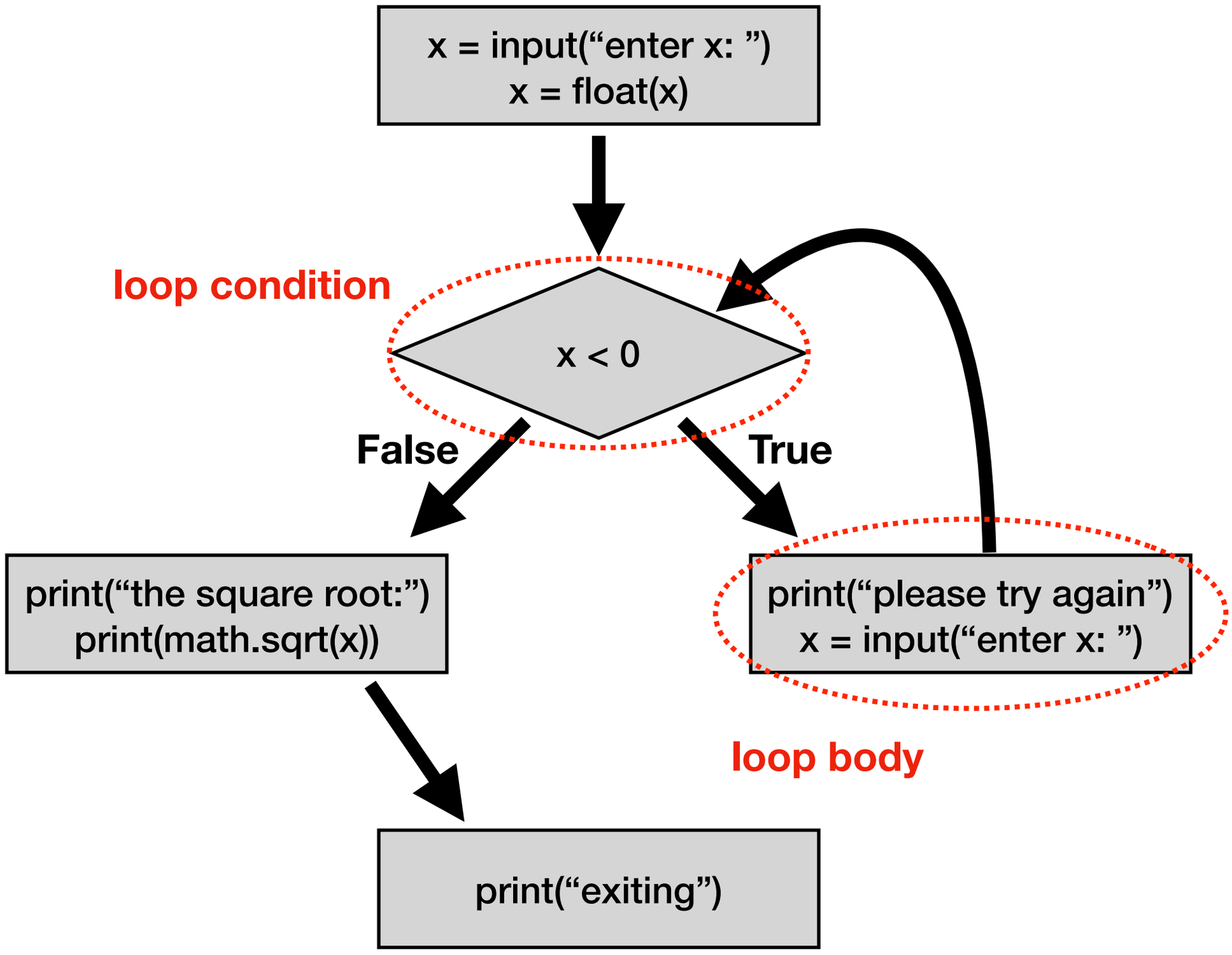


Control Flow Diagrams: “while”

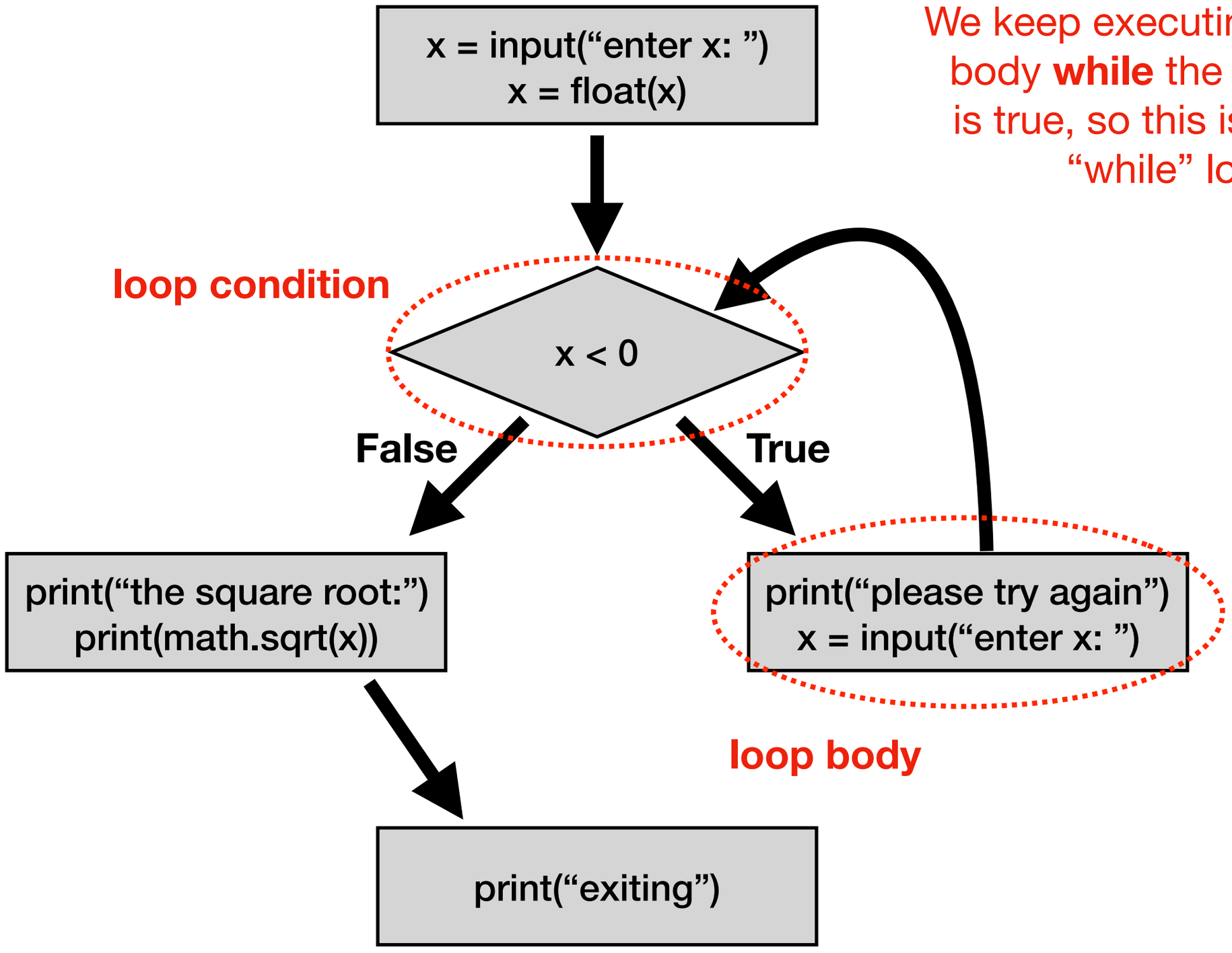


Each time through is called an “iteration”

Control Flow Diagrams: “while”

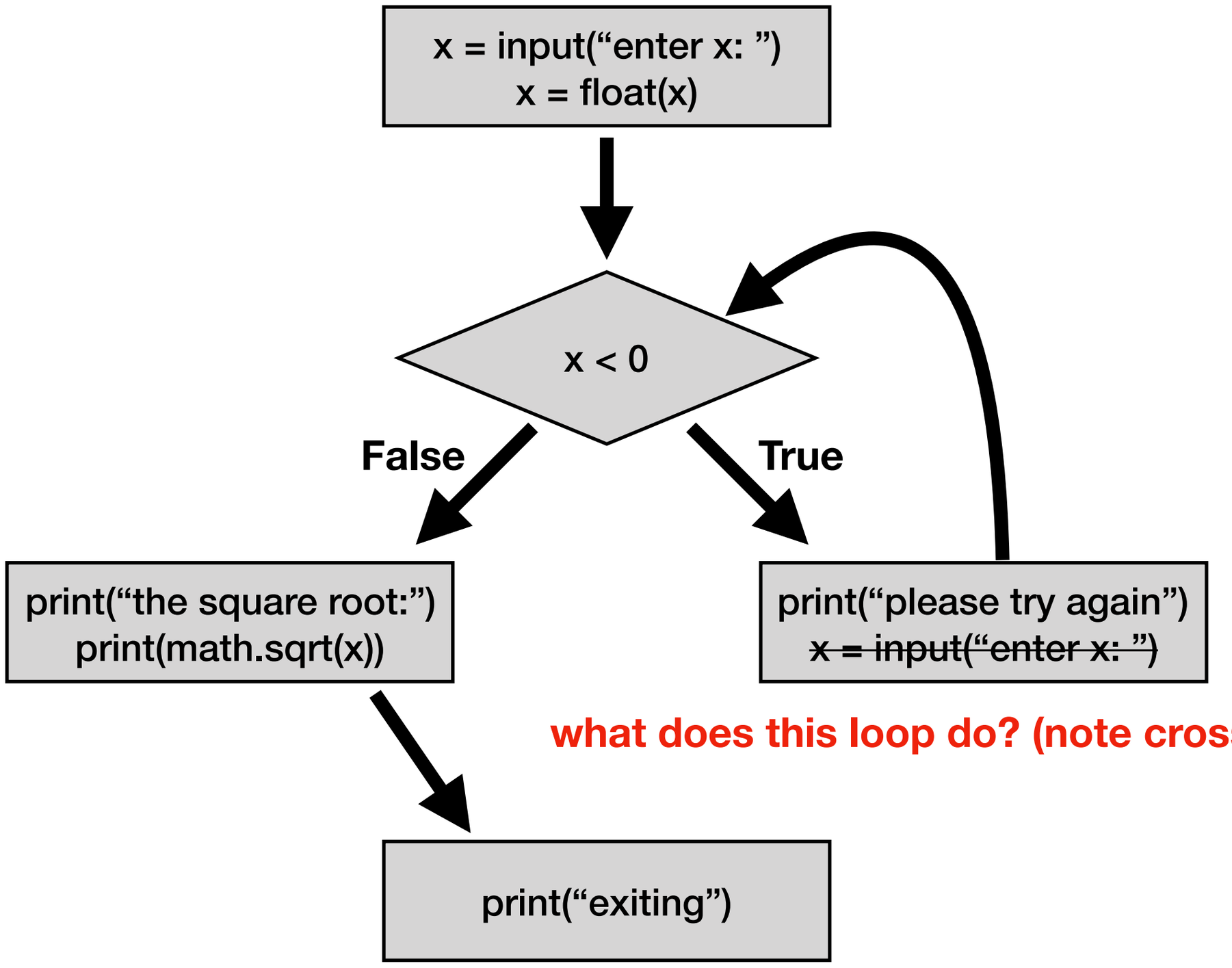


Control Flow Diagrams: “while”



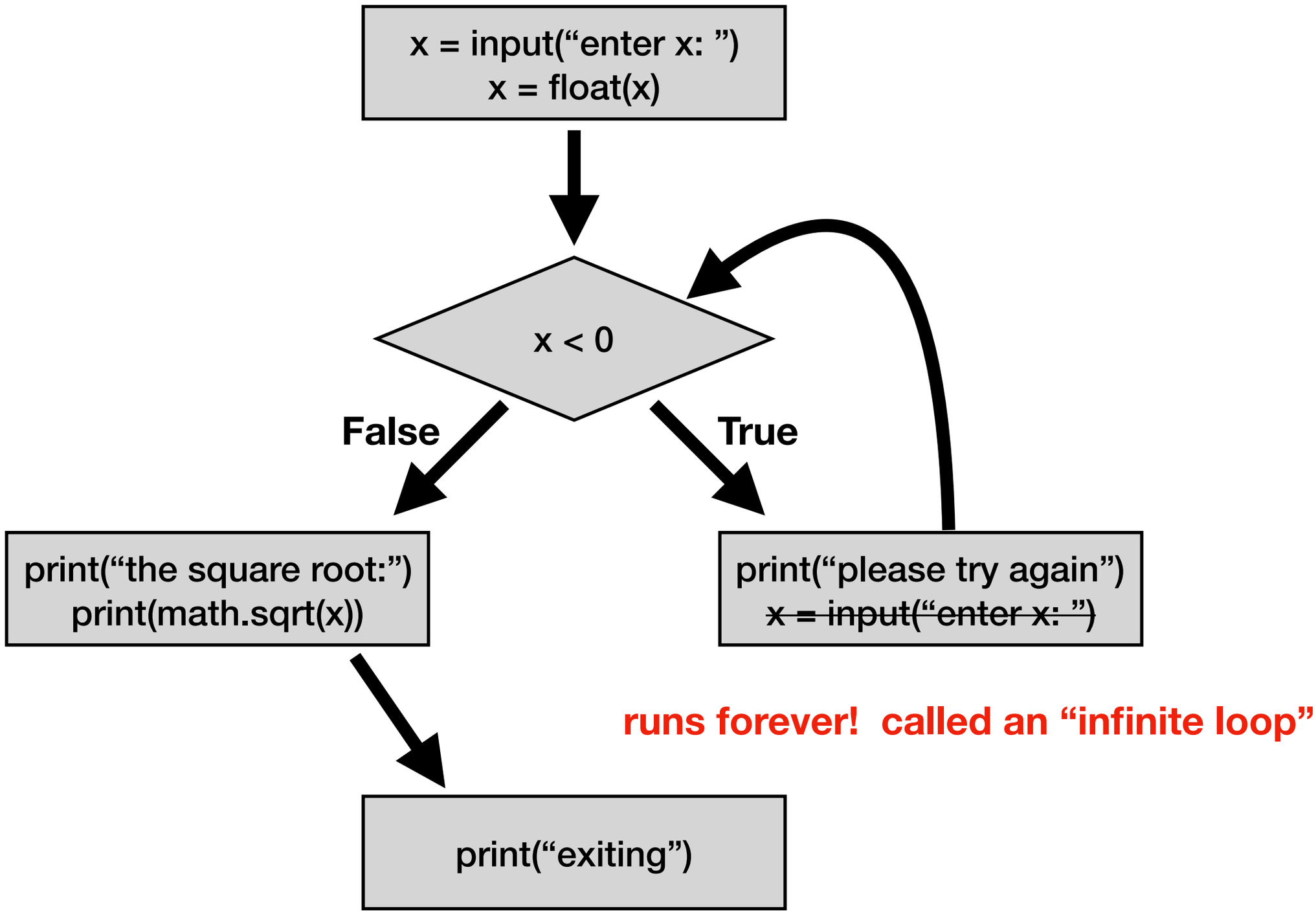
We keep executing the loop body **while** the condition is true, so this is called a “while” loop

Control Flow Diagrams: “while”



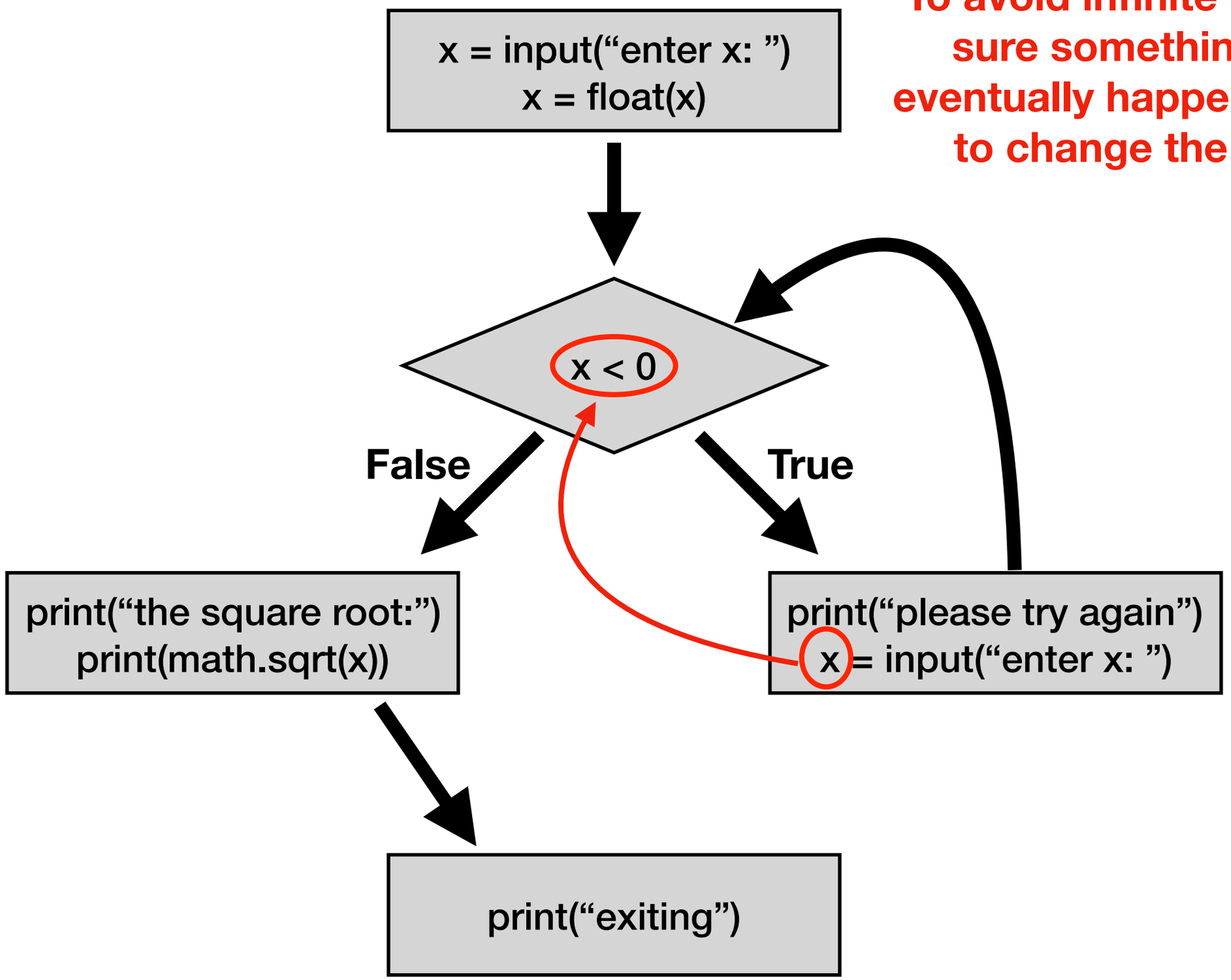
what does this loop do? (note crossed out line)

Control Flow Diagrams: “while”



runs forever! called an “infinite loop”

Control Flow Diagrams: “while”



To avoid infinite loops, make sure something will/can eventually happen in the body to change the condition

Today's Outline

Control Flow Diagrams

Basic syntax for “while”

Demos

Syntax

```
x = int(input(“enter x: “))

if x < 0:
    x = int(input(“please try again: ”))
```

Syntax for “if”

Syntax

```
x = int(input(“enter x: “))

if x < 0:
    x = int(input(“please try again: ”))
```

Syntax for “if”

Syntax

```
x = int(input(“enter x: “))

while x < 0:
    x = int(input(“please try again: ”))
```

Syntax for “while loop” is just like for “if”, just replace “if” with “while”

Syntax

```
x = int(input(“enter x: “))

while x < 0:
    x = int(input(“please try again: ”))
```

this example gives user an arbitrary number of tries until they get it right

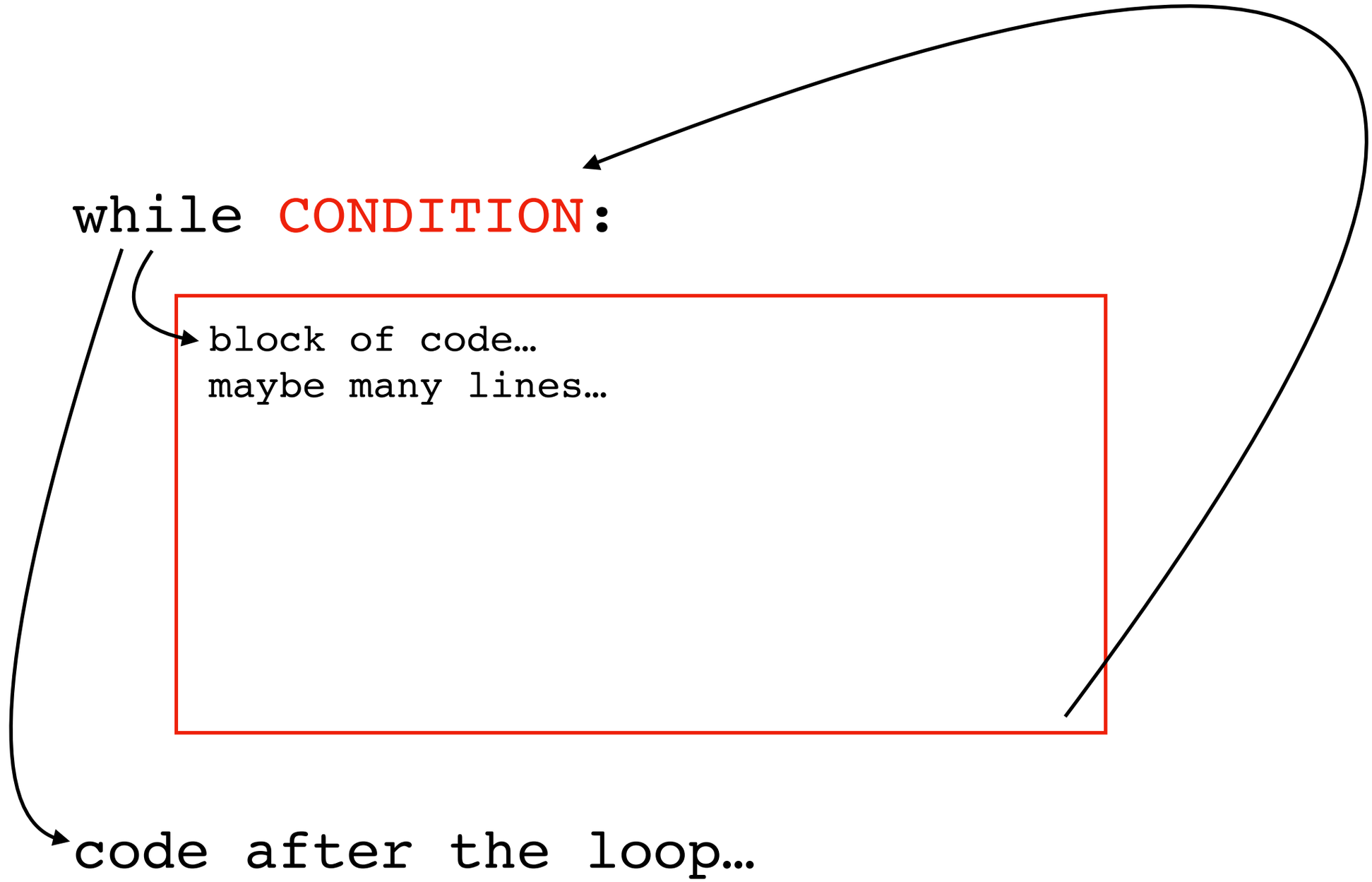
Control Flow

```
while CONDITION:
    # your code
```

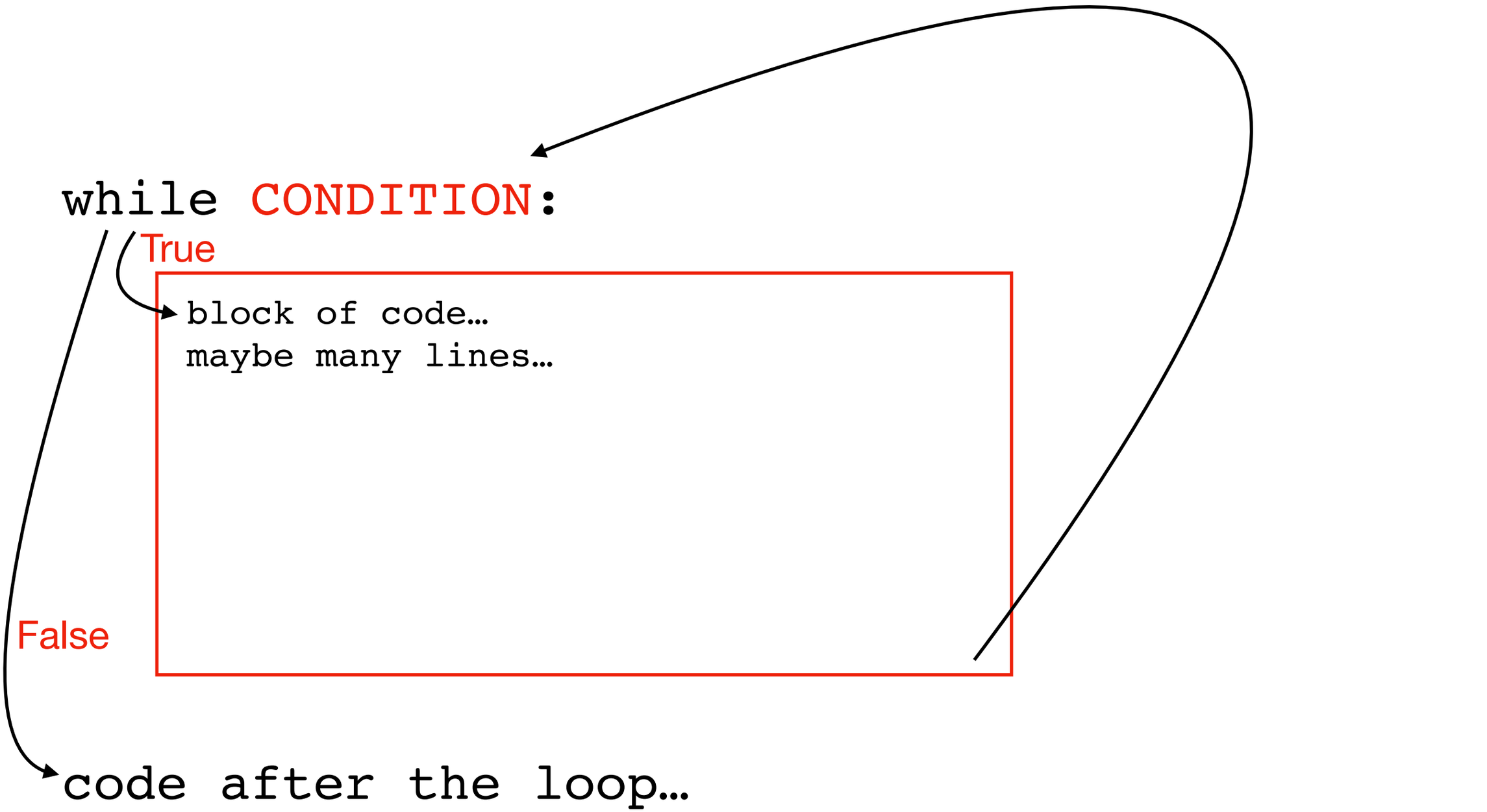
Control Flow

```
while CONDITION:
    block of code…
    maybe many lines…

code after the loop…
```

Control Flow



Control Flow



Congrats!

You now understand the 4 key **Flow of Execution** ideas,
in the context of Python.

1. **generally, proceed forward, one step at a time**

2. sometimes go run a “mini program” somewhere else before continuing to the next line
   - This is a function call

3. sometimes skip forward over some lines of code
   - Conditional or while loop, when the condition is false

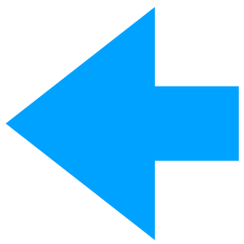
4. sometimes go back to a previous line of code
   - while loop. When at the end of body, always go back to condition

three primary exceptions to the general case (1)

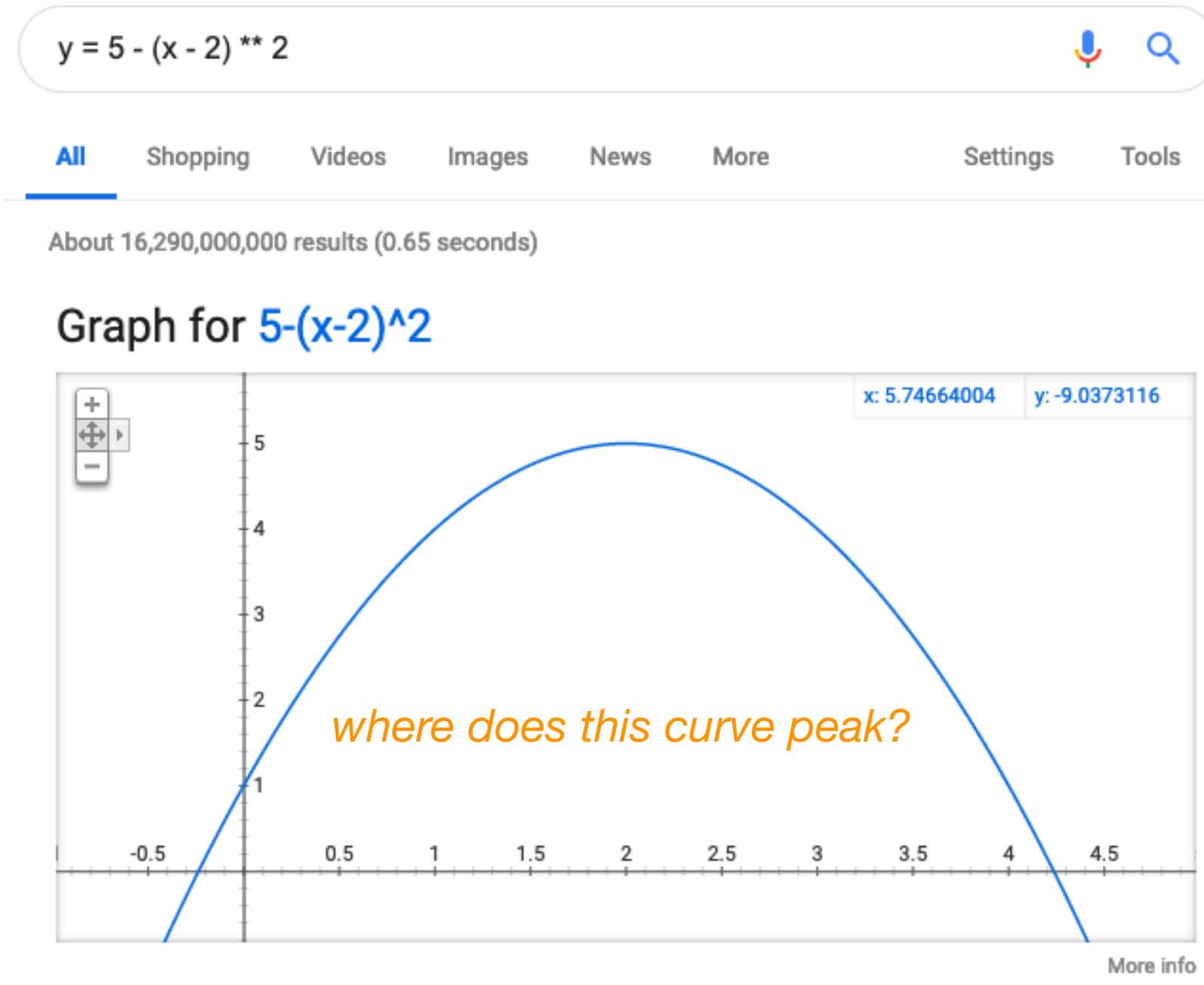
Today's Outline

Control Flow Diagrams

Basic syntax for “while”

Demos 

Demo: Maximum (Finding the Peak)



Demo: Countdown Timer

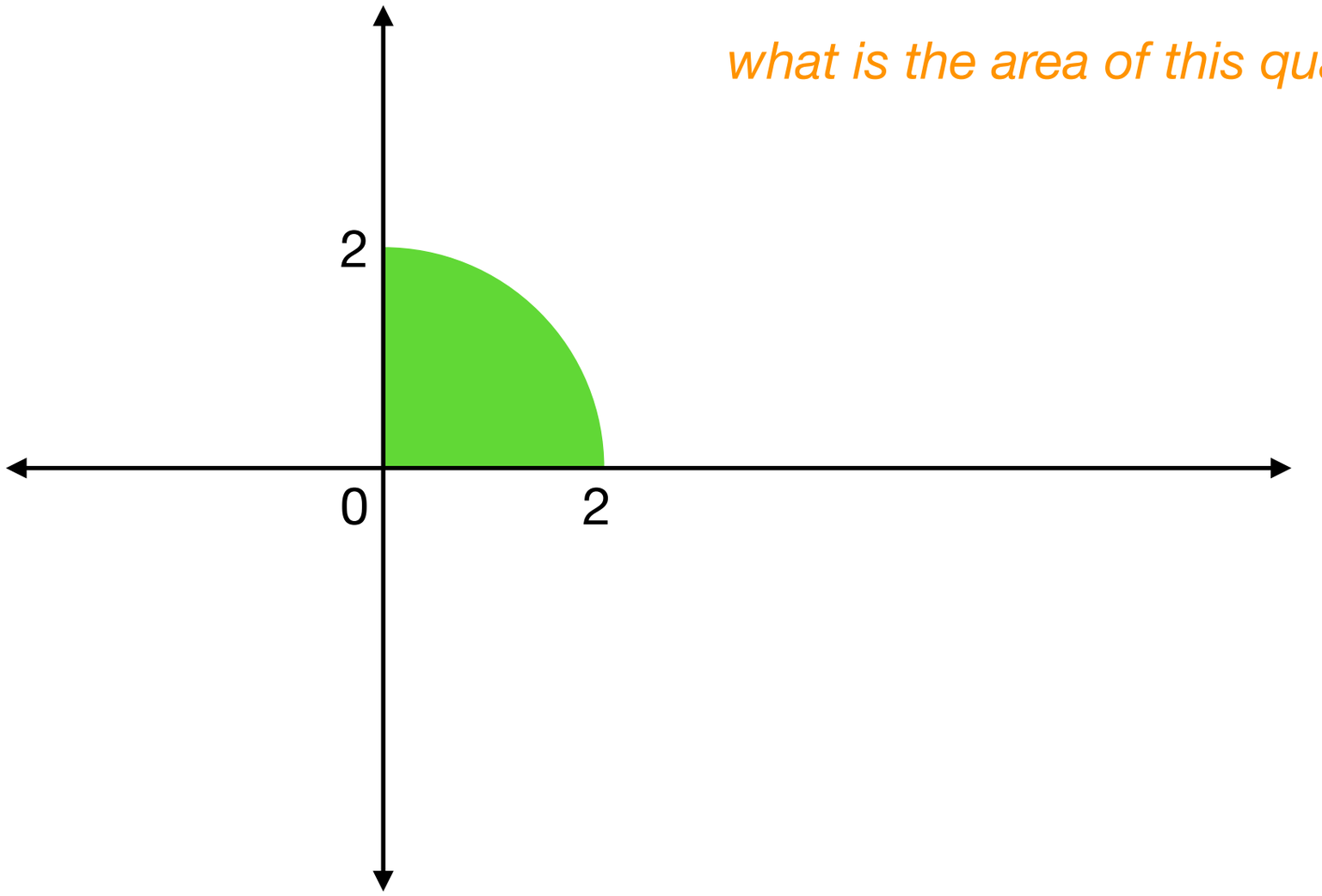
use `time.sleep(1)`

```
how many seconds? 5
5
4
3
2
1
DING DING DING DING DING!
how many seconds? 2
2
1
0
DING DING DING DING DING!
how many seconds? q
good bye!
```

exit program

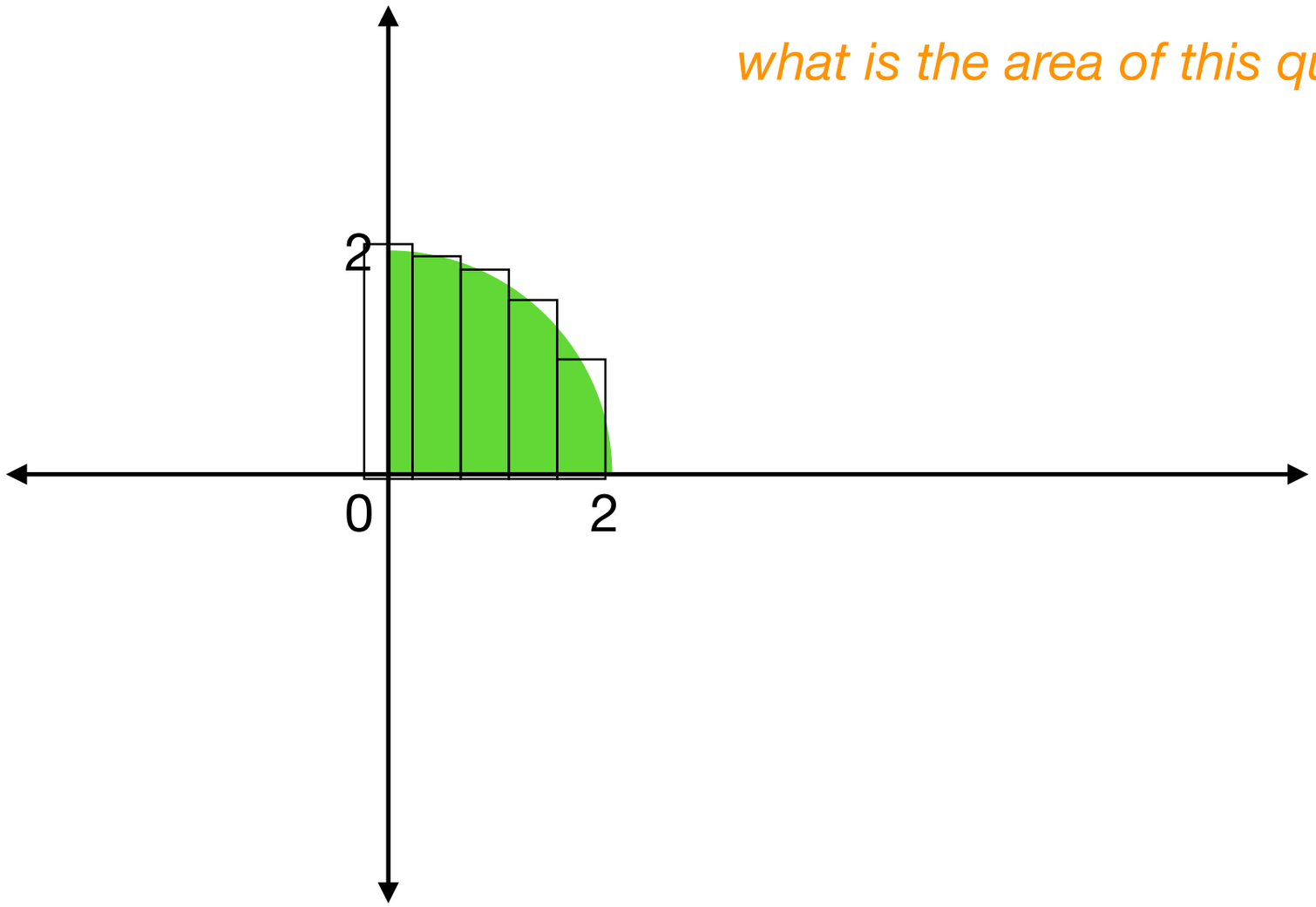
this program will involve a nested loop!!!

Demo: Integration (Finding the area)



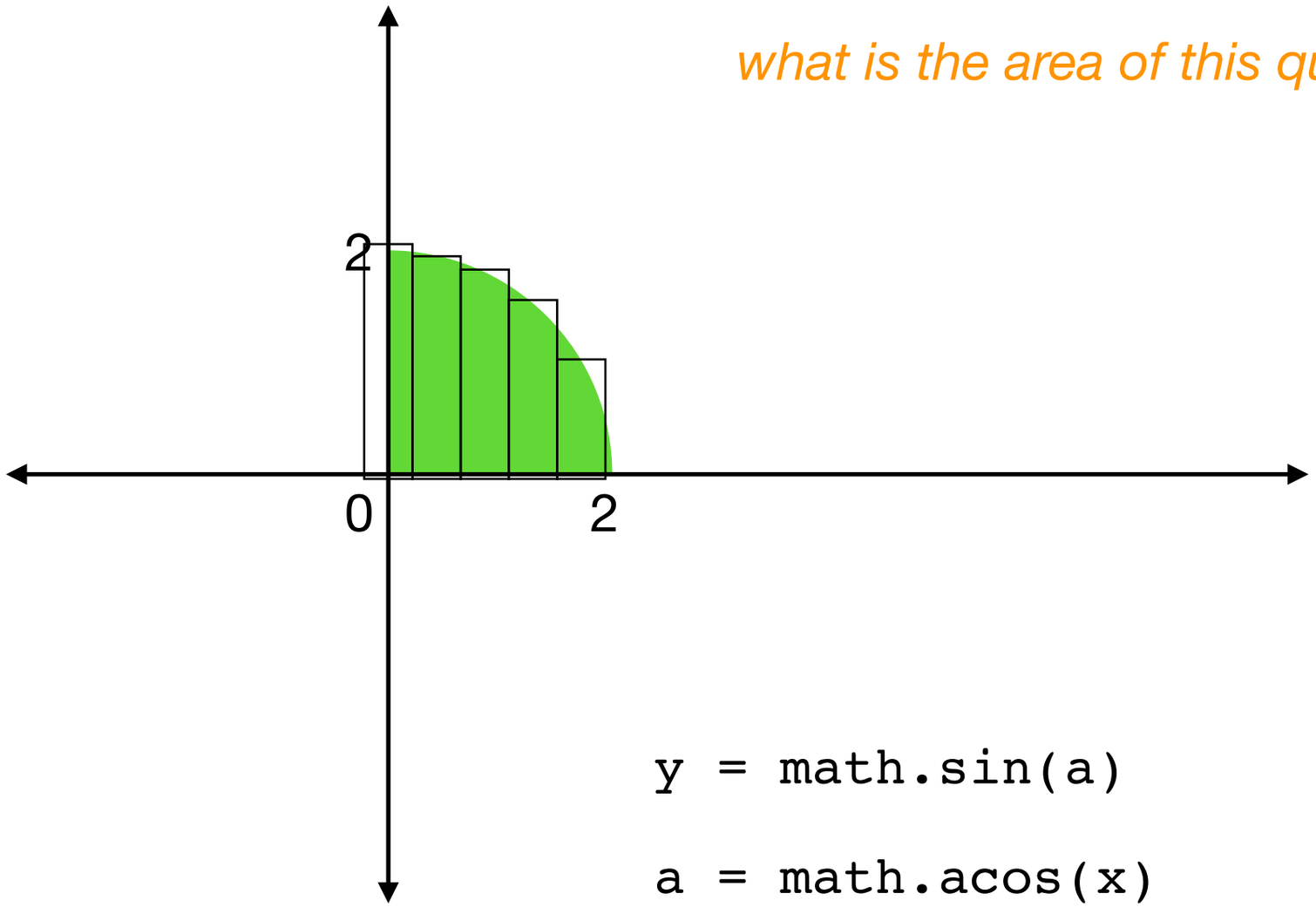
what is the area of this quarter circle?

Demo: Integration (Finding the area)



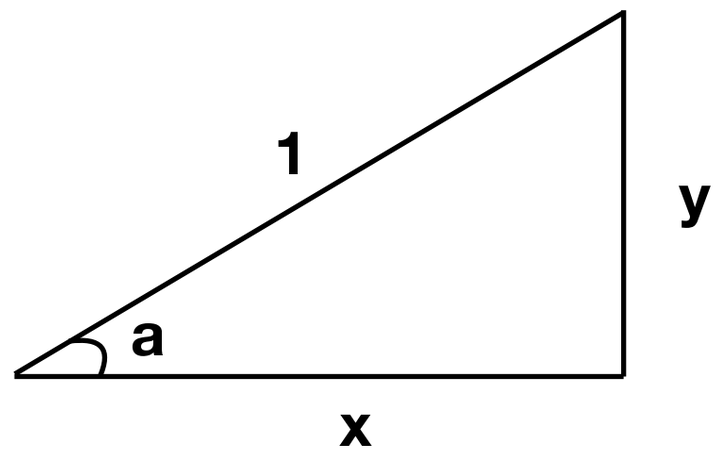
what is the area of this quarter circle?

Demo: Integration (Finding the area)

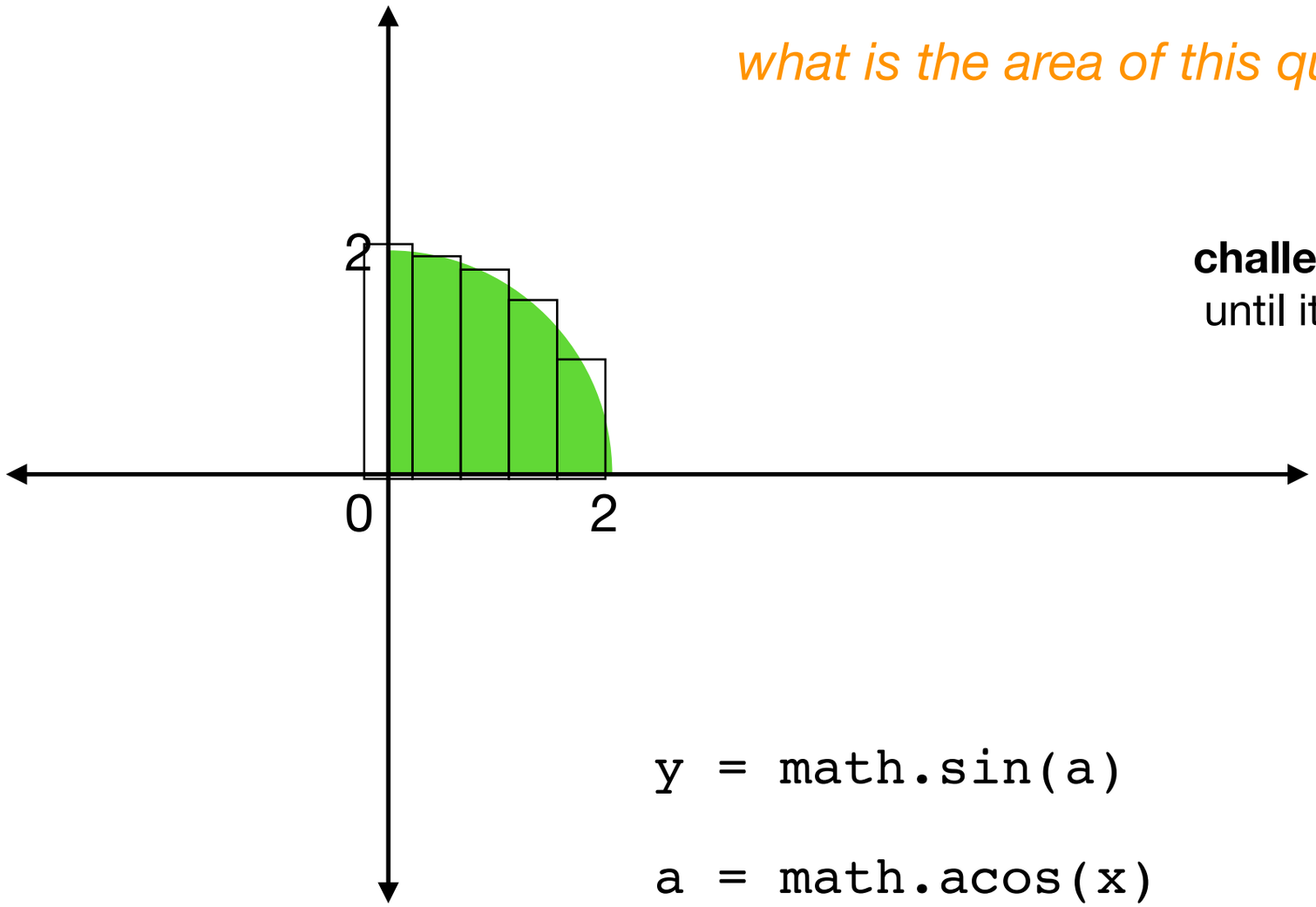


what is the area of this quarter circle?

```
y = math.sin(a)

a = math.acos(x)
```



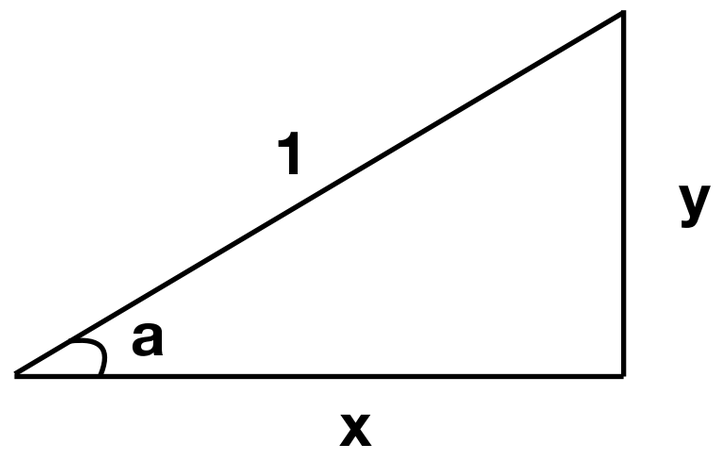
Demo: Integration (Finding the area)



what is the area of this quarter circle?

challenge: keep using narrower slices until it doesn't make much difference

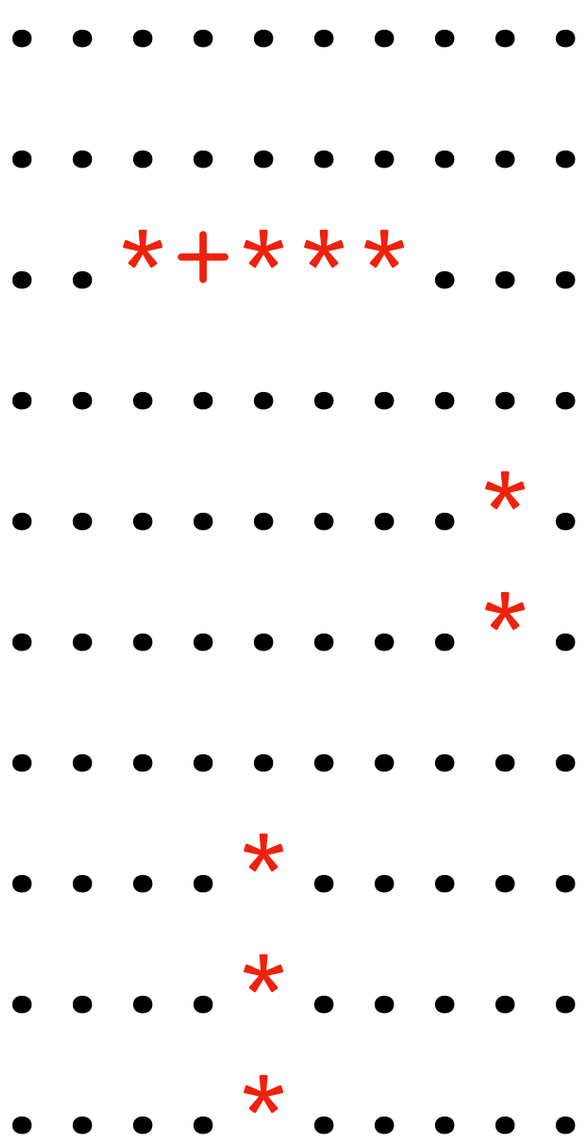
```
y = math.sin(a)

a = math.acos(x)
```



Demo: Prime Finder

```
Here are a "few" primes:
2
3
5
7
11
13
... runs forever ...
```

Demo: Battleship



show where ship(s) are after guess

```
guess and ship: +
     just ship: *
guess and miss: -
    blank spot: .
```