[301] Iteration

Tyler Caraza-Harter

Review

```
n = 10
if n > 1:
    print('over 1')
    if n > 2:
        print('over 2')
        if n > 3:
            print('over 3')
            if n > 4:
                print('over 4')
        print('hi')
```

What does it print?

Review

```
n = 10
if n > 1:
    print('over 1')
    if n > 2:
        print('over 2')
        if n > 3:
            print('over 3')
            if n > 4:
                print('over 4')
        print('hi')
```

What does it print?

```
over 1
over 2
over 3
over 4
hi
```

Review

```
n = 10
if n > 1:
    print('over 1')
    if n > 2:
        print('over 2')
        if n > 3:
            print('over 3')
            if n > 4:
                print('over 4')
        print('hi')
```

What is the smallest integer value we could change n to at the beginning and still have it print “hi”?

A: 2
B: 3
C: 4
D: 5

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

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

Learn to avoid pitfalls

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

Worksheet

State:

6

N 4

total 0

answer 0

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:

6

N 4

total 0

answer 0

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

6

State:

N 4

total 0

answer 0

loop condition

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 body

Worksheet

State:

6

N 4

total 0

answer 0

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

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

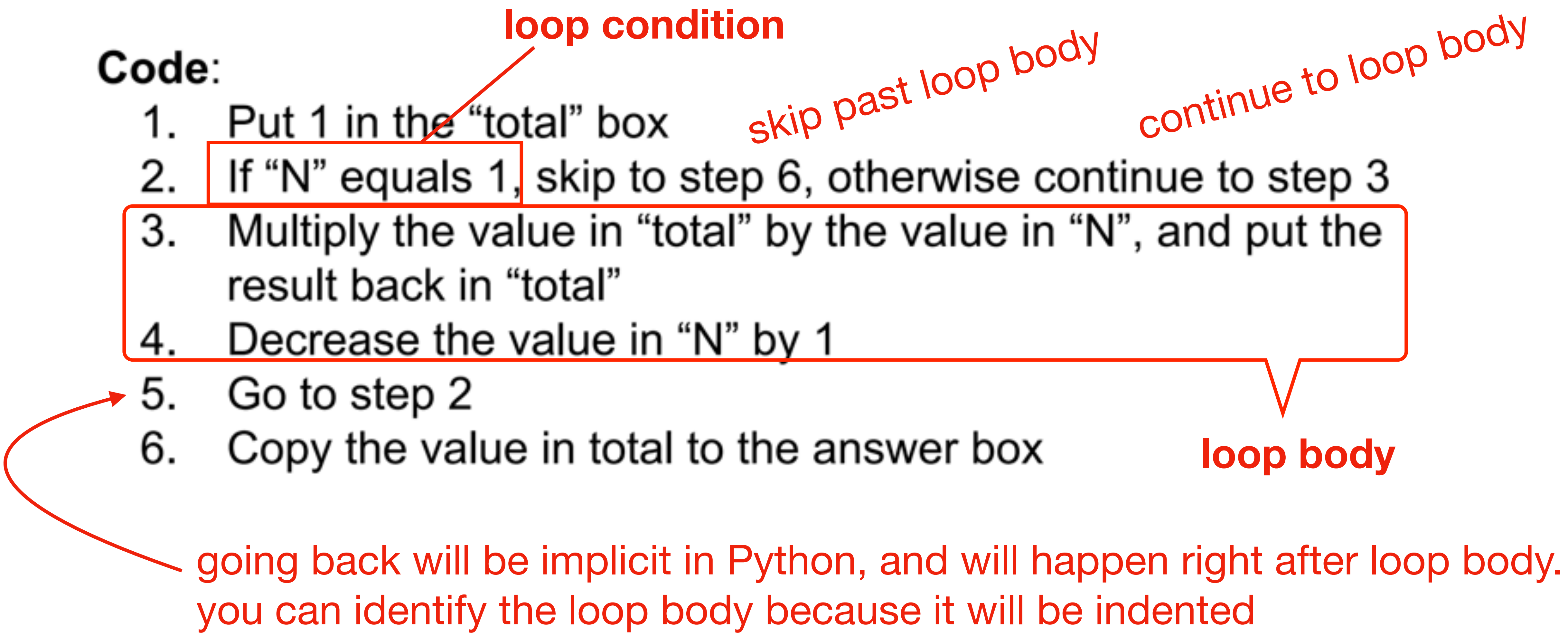
State:

6

N 4

total 0

answer 0



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

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

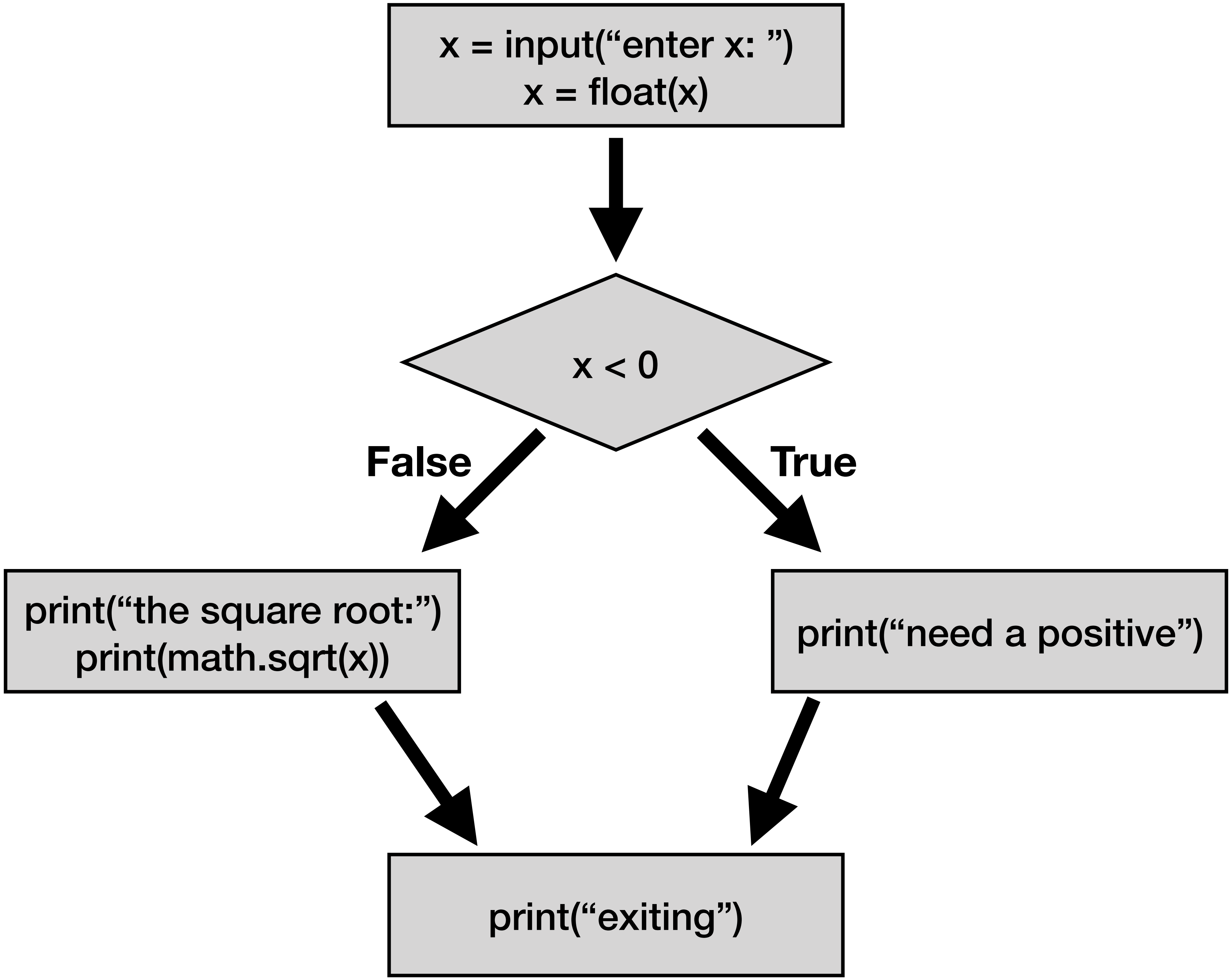
Today's Outline

Control Flow Diagrams

Basic syntax for “while”

Demos

Control Flow Diagrams: “if”



x = input(“enter x: ”)
x = float(x)

x < 0

False

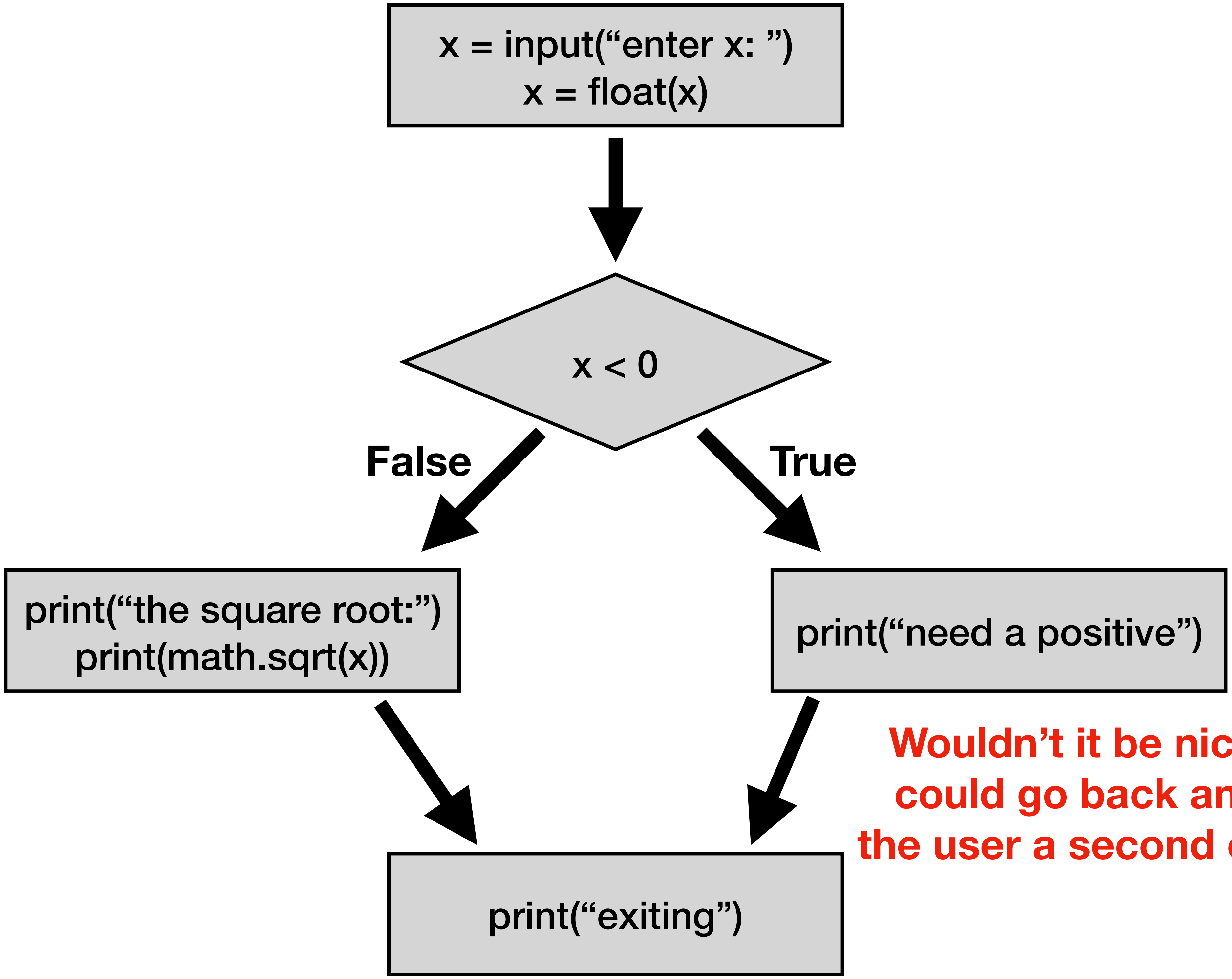
True

print(“the square root:”)
print(math.sqrt(x))

print(“need a positive”)

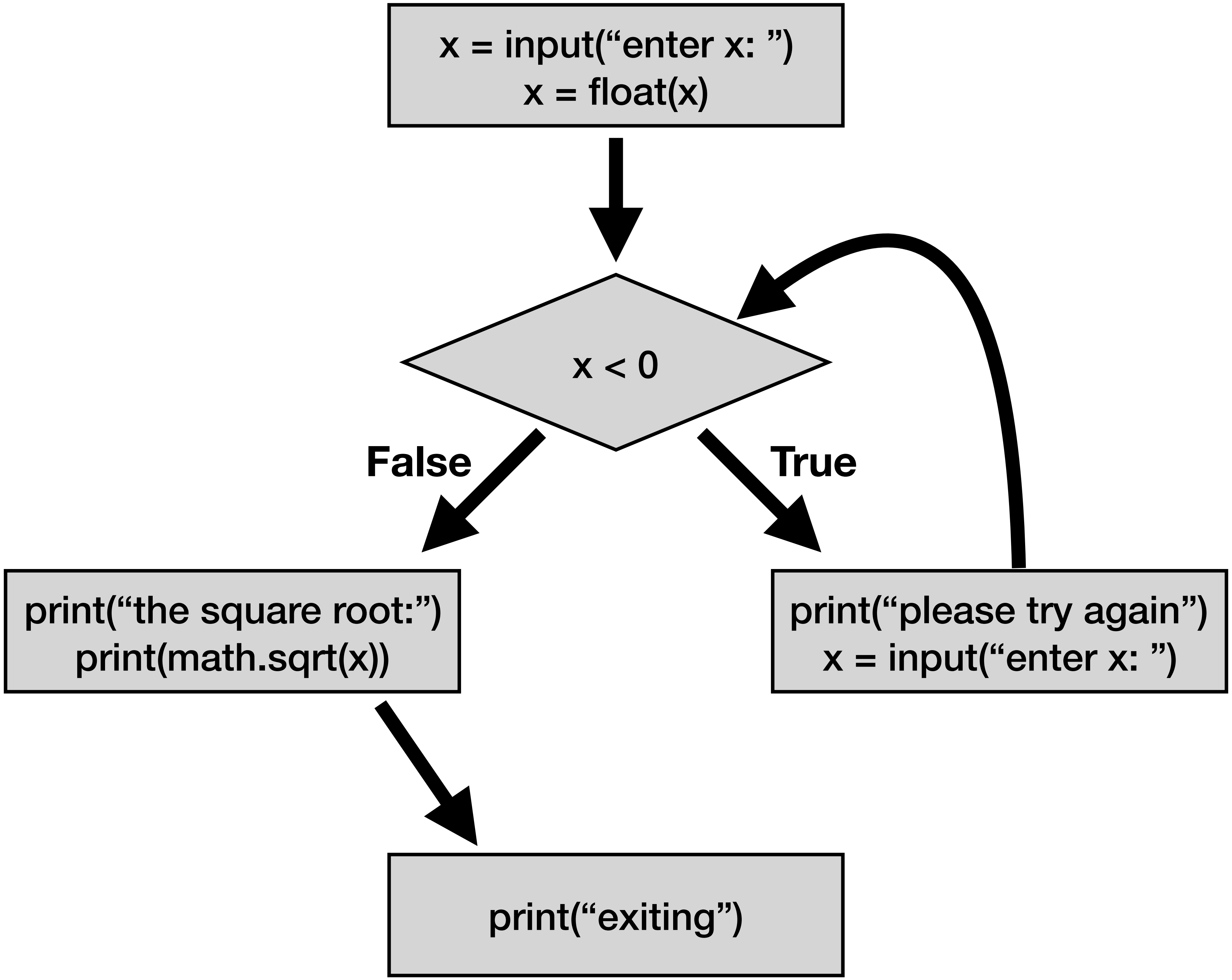
print(“exiting”)

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”



x = input(“enter x: ”)
x = float(x)

x < 0

False

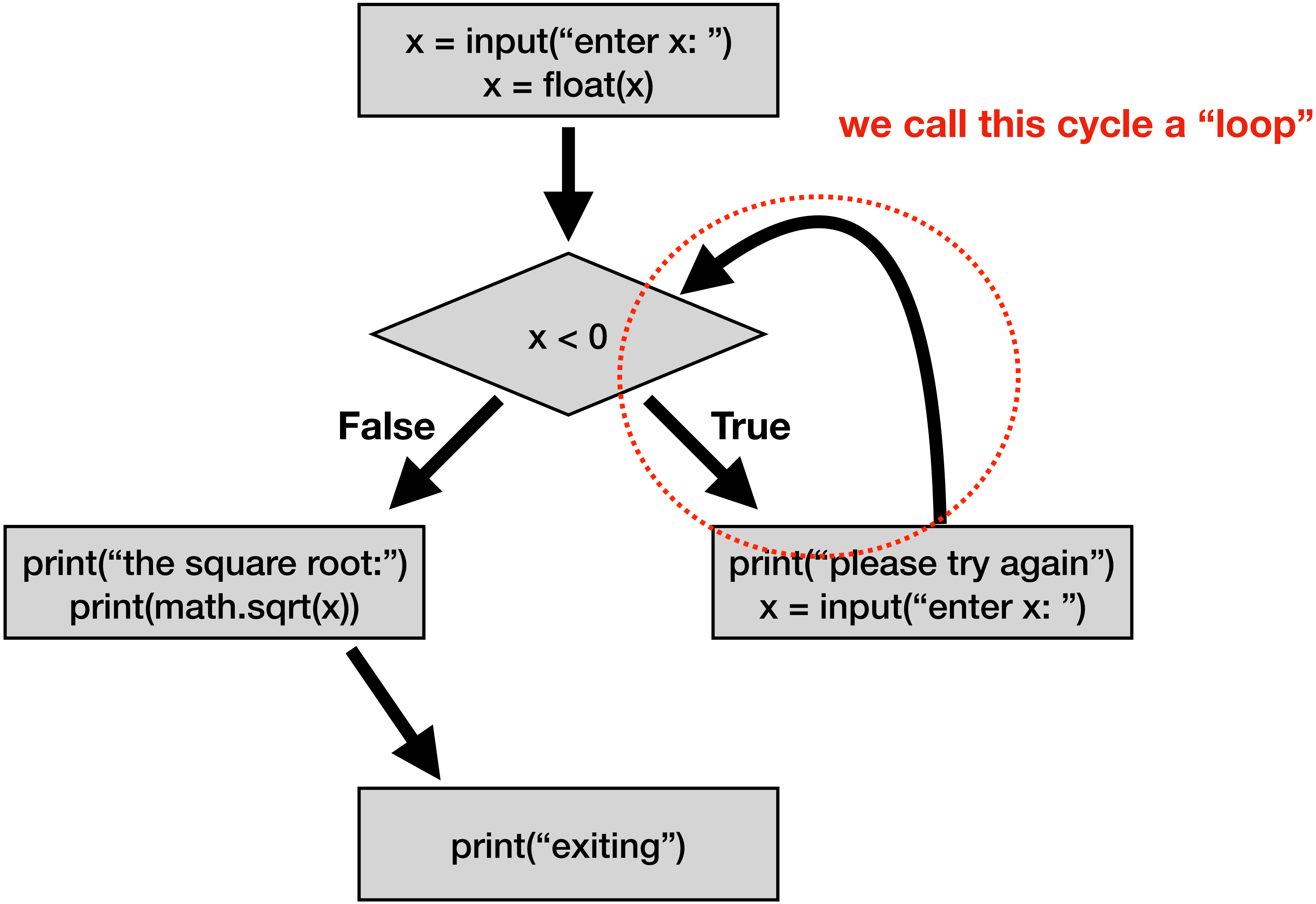
True

print(“the square root:”)
print(math.sqrt(x))

print(“please try again”)
x = input(“enter x: ”)

print(“exiting”)

Control Flow Diagrams: “while”



x = input(“enter x: ”)
x = float(x)

we call this cycle a “loop”

x < 0

False

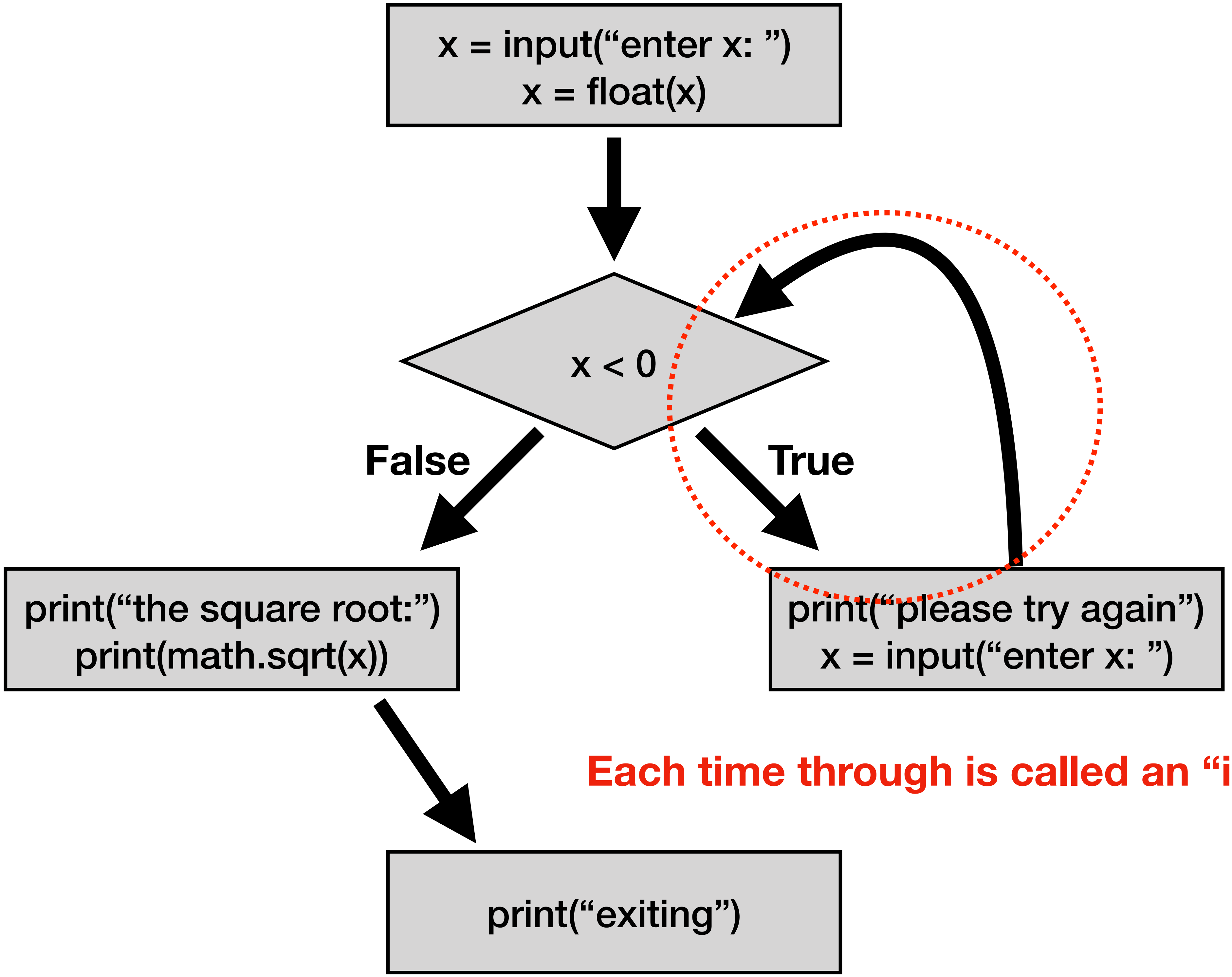
True

print(“the square root:”)
print(math.sqrt(x))

print(“please try again”)
x = input(“enter x: ”)

print(“exiting”)

Control Flow Diagrams: “while”



x = input(“enter x: ”)
x = float(x)

x < 0

False

True

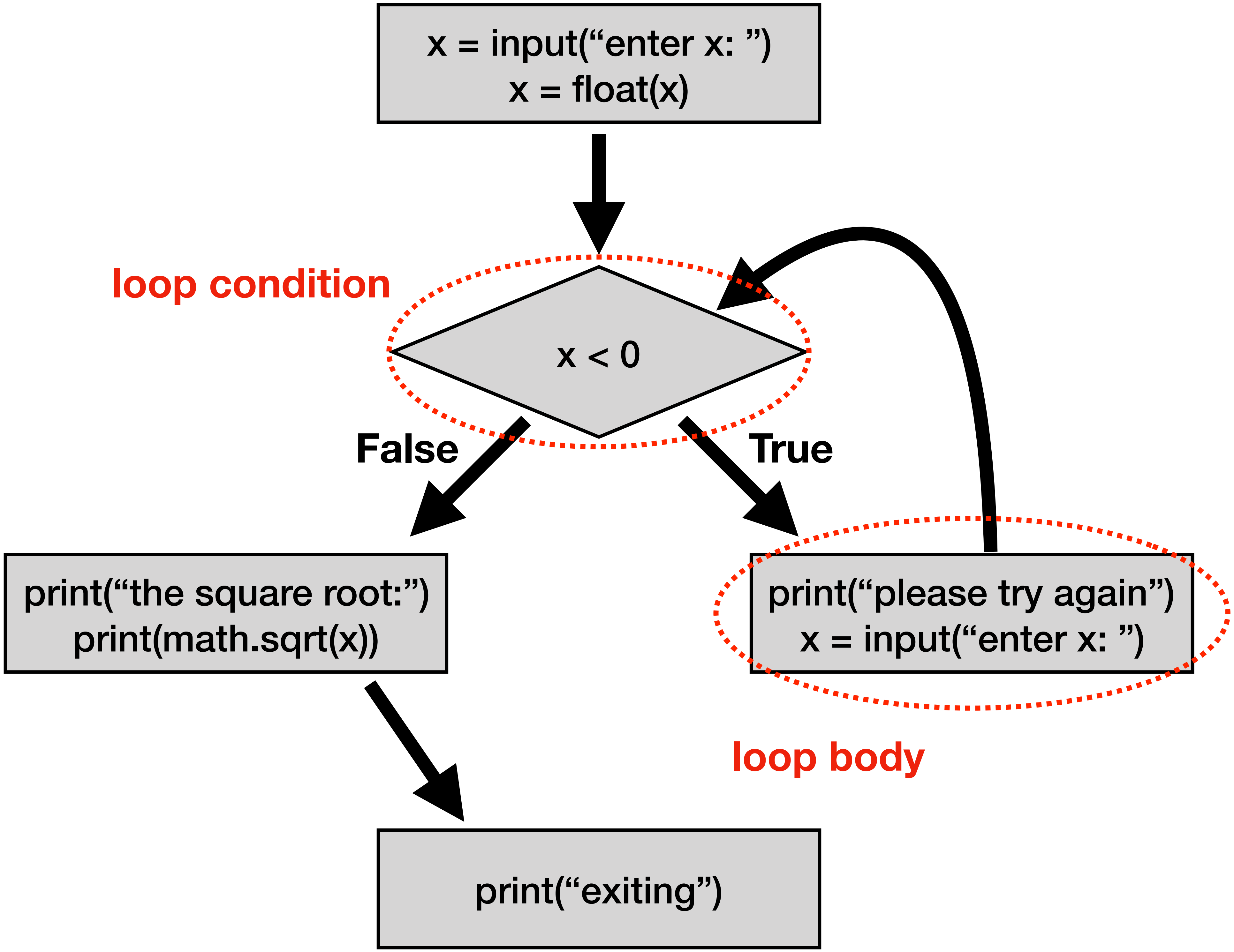
print(“the square root:”)
print(math.sqrt(x))

print(“please try again”)
x = input(“enter x: ”)

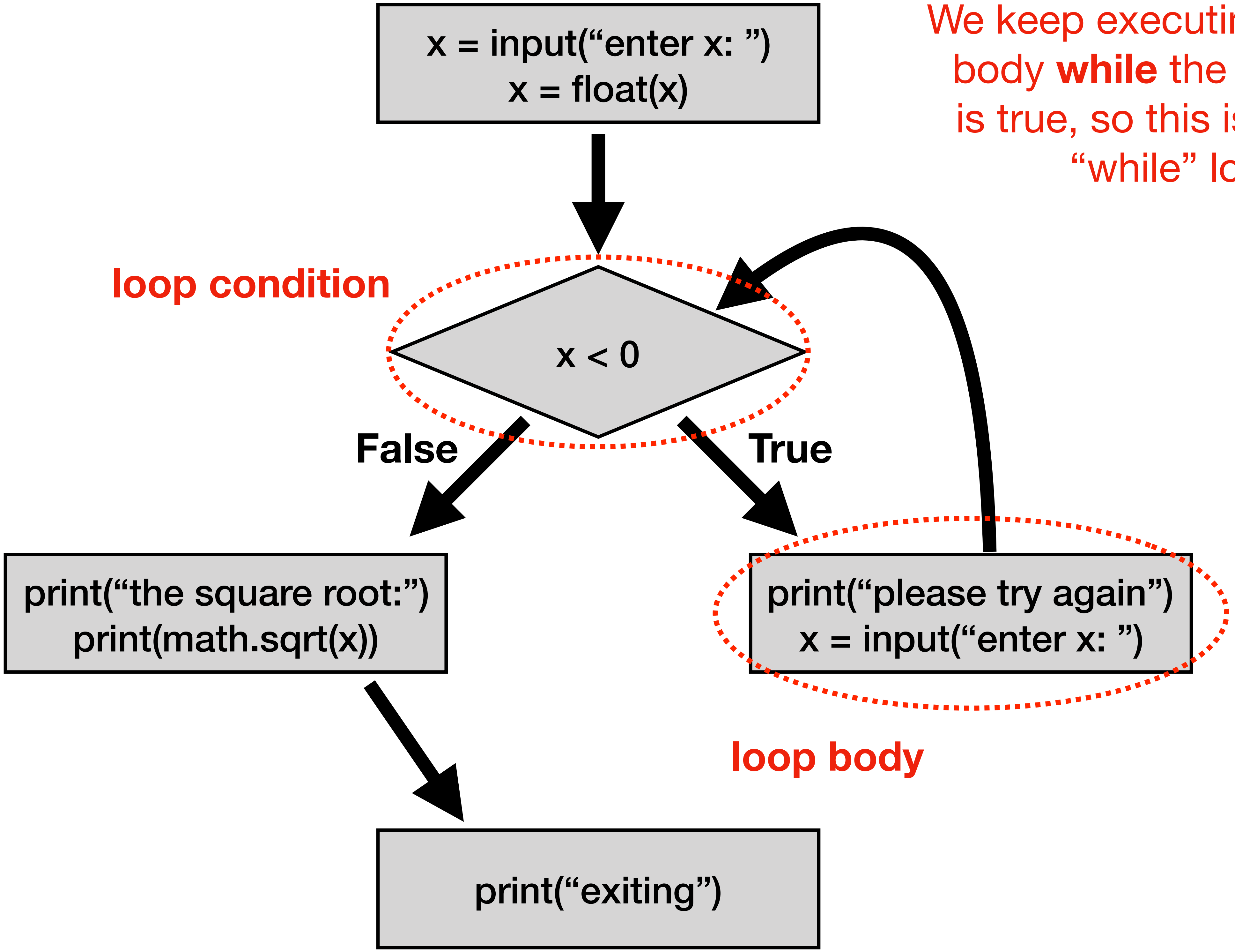
print(“exiting”)

Each time through is called an “iteration”

Control Flow Diagrams: “while”

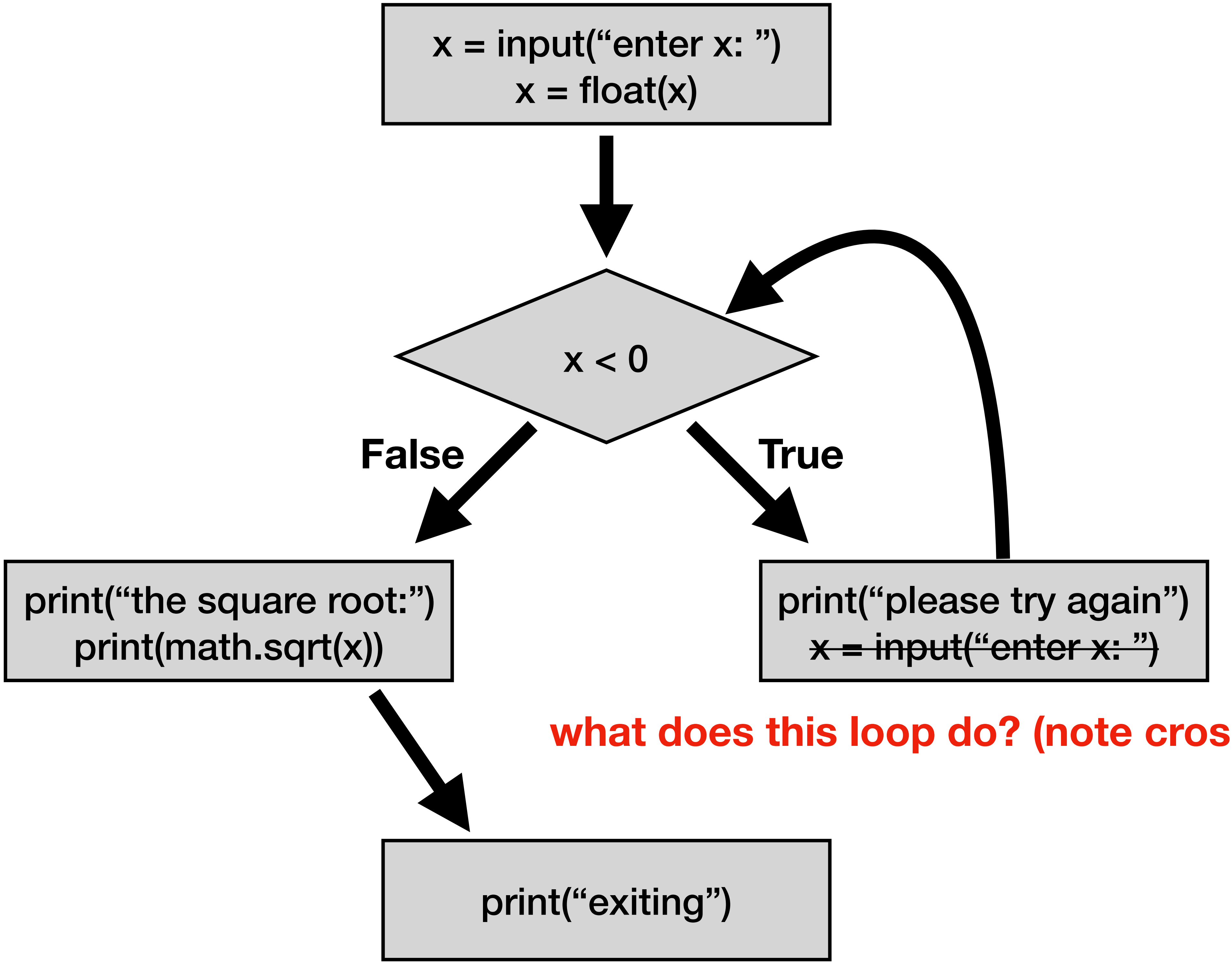


Control Flow Diagrams: “while”



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

Control Flow Diagrams: “while”



x = input(“enter x: ”)
x = float(x)

x < 0

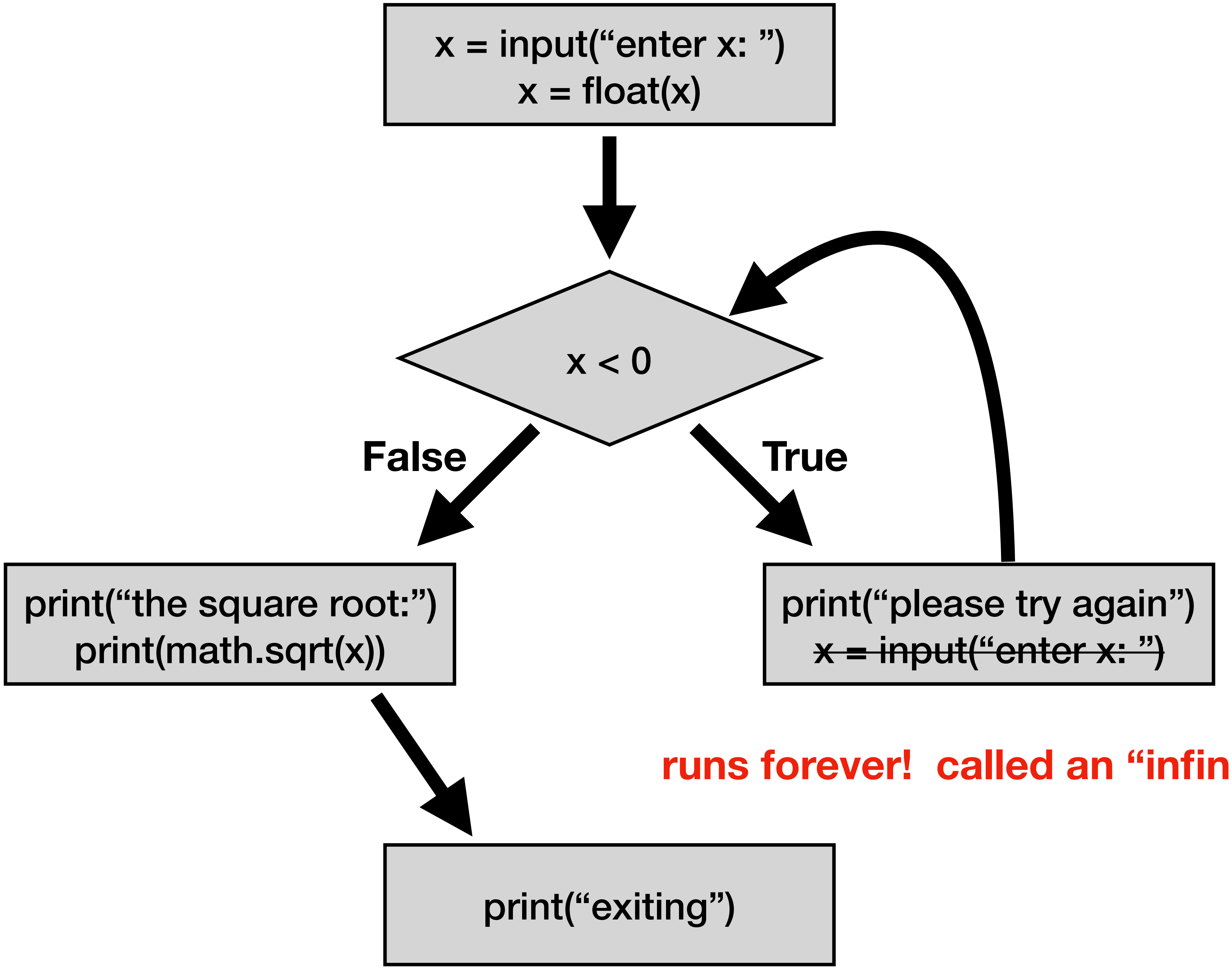
print(“the square root:”)
print(math.sqrt(x))

print(“please try again”)
~~x = input(“enter x: ”)~~

print(“exiting”)

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

Control Flow Diagrams: “while”



x = input(“enter x: ”)
x = float(x)

x < 0

False

True

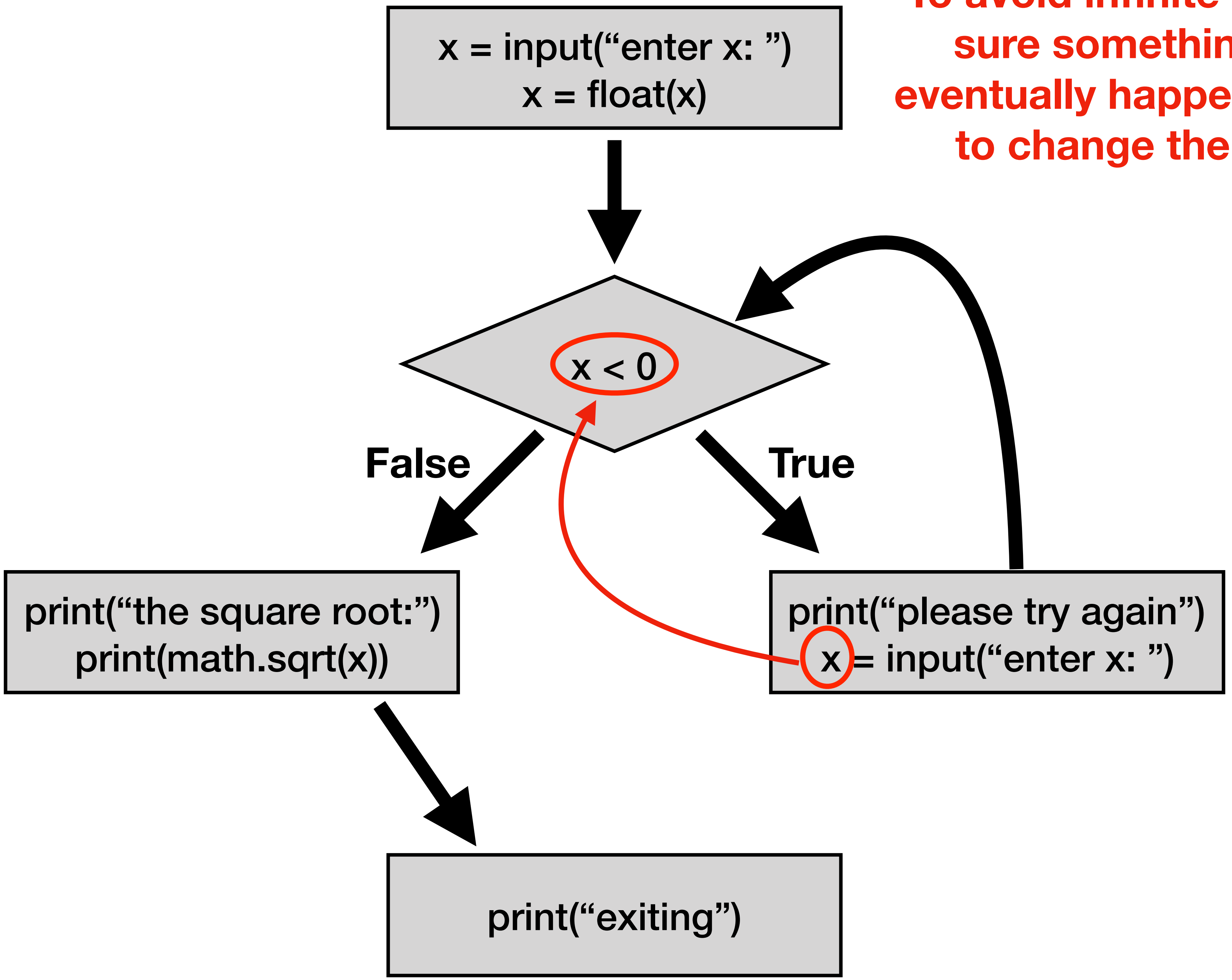
print(“the square root:”)
print(math.sqrt(x))

print(“please try again”)
~~x = input(“enter x: ”)~~

print(“exiting”)

runs forever! called an “infinite loop”

Control Flow Diagrams: “while”



x = input(“enter x: ”)
x = float(x)

x < 0

False

True

print(“the square root:”)
print(math.sqrt(x))

print(“please try again”)
x = input(“enter x: ”)

print(“exiting”)

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

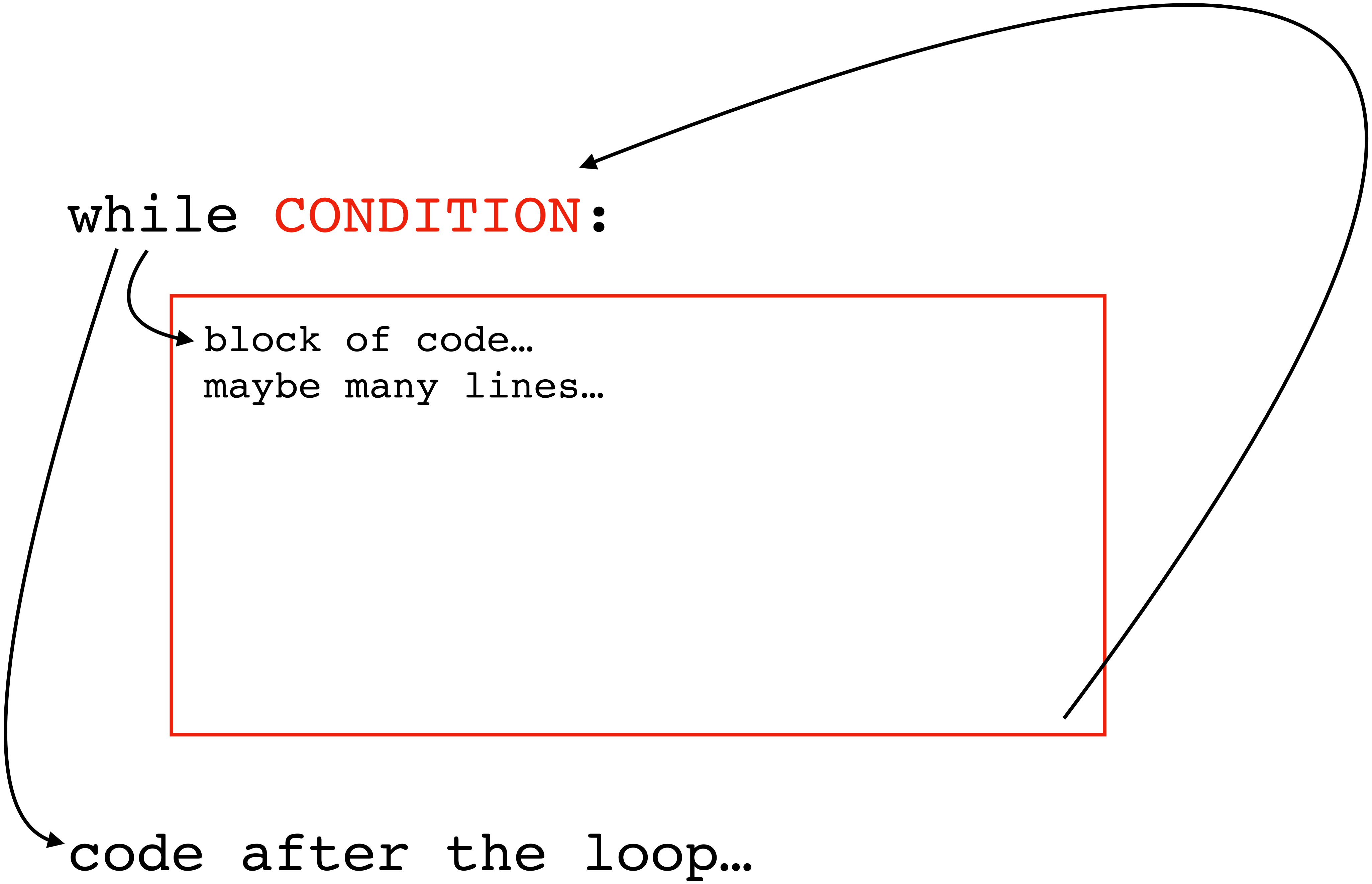
Control Flow

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

Control Flow

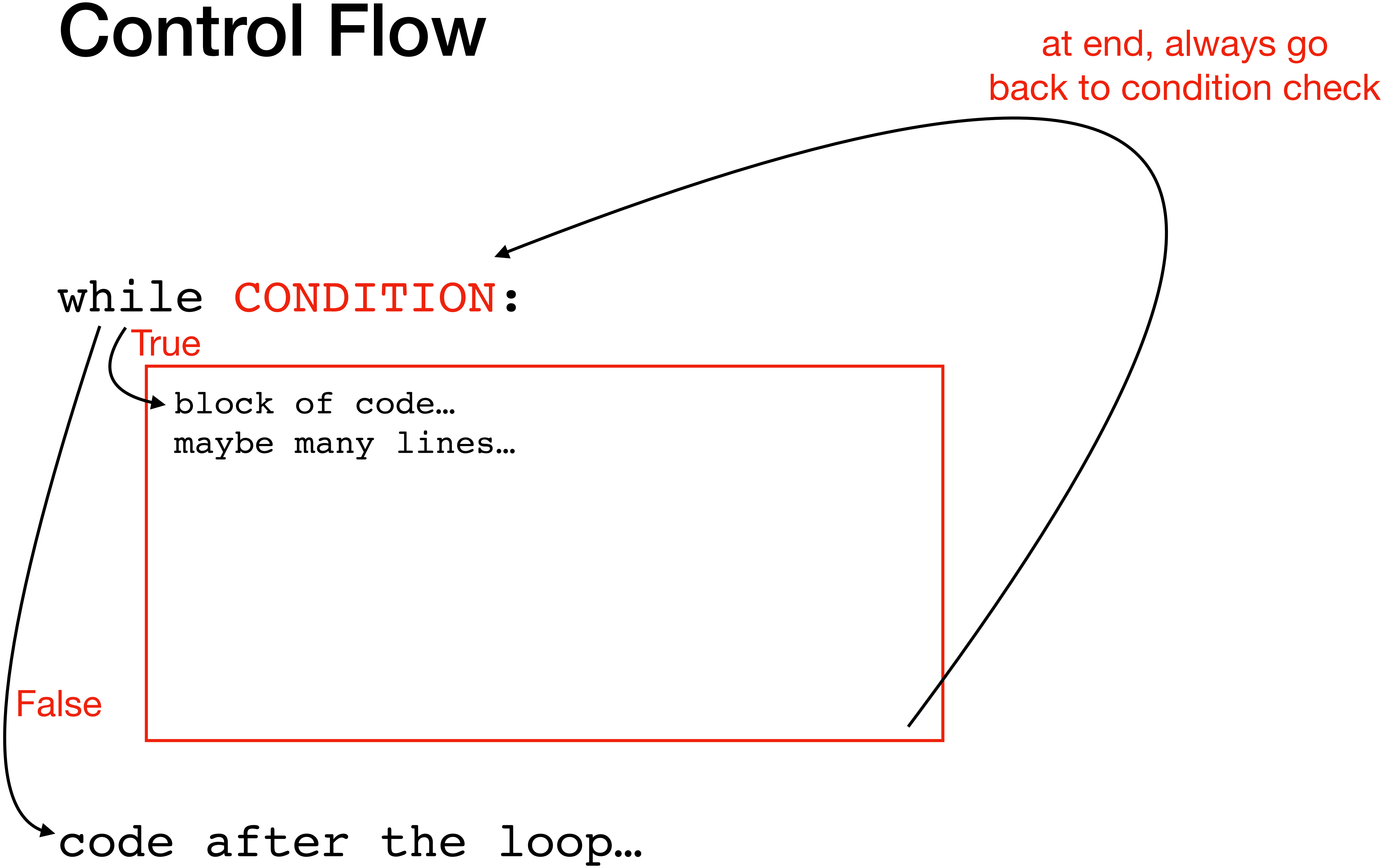
```
while CONDITION:

    block of code…
    maybe many lines…

code after the loop…
```

Control Flow



```
while CONDITION:
    block of code…
    maybe many lines…
code after the loop…
```

Control Flow



at end, always go back to condition check

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

code after the loop…
```

True

False

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