

[220] Conditionals

Department of Computer Sciences
University of Wisconsin-Madison

Readings:

Parts of Chapter 5 of Think Python
Chapter 5.5 to 5.8 of Python for Everybody

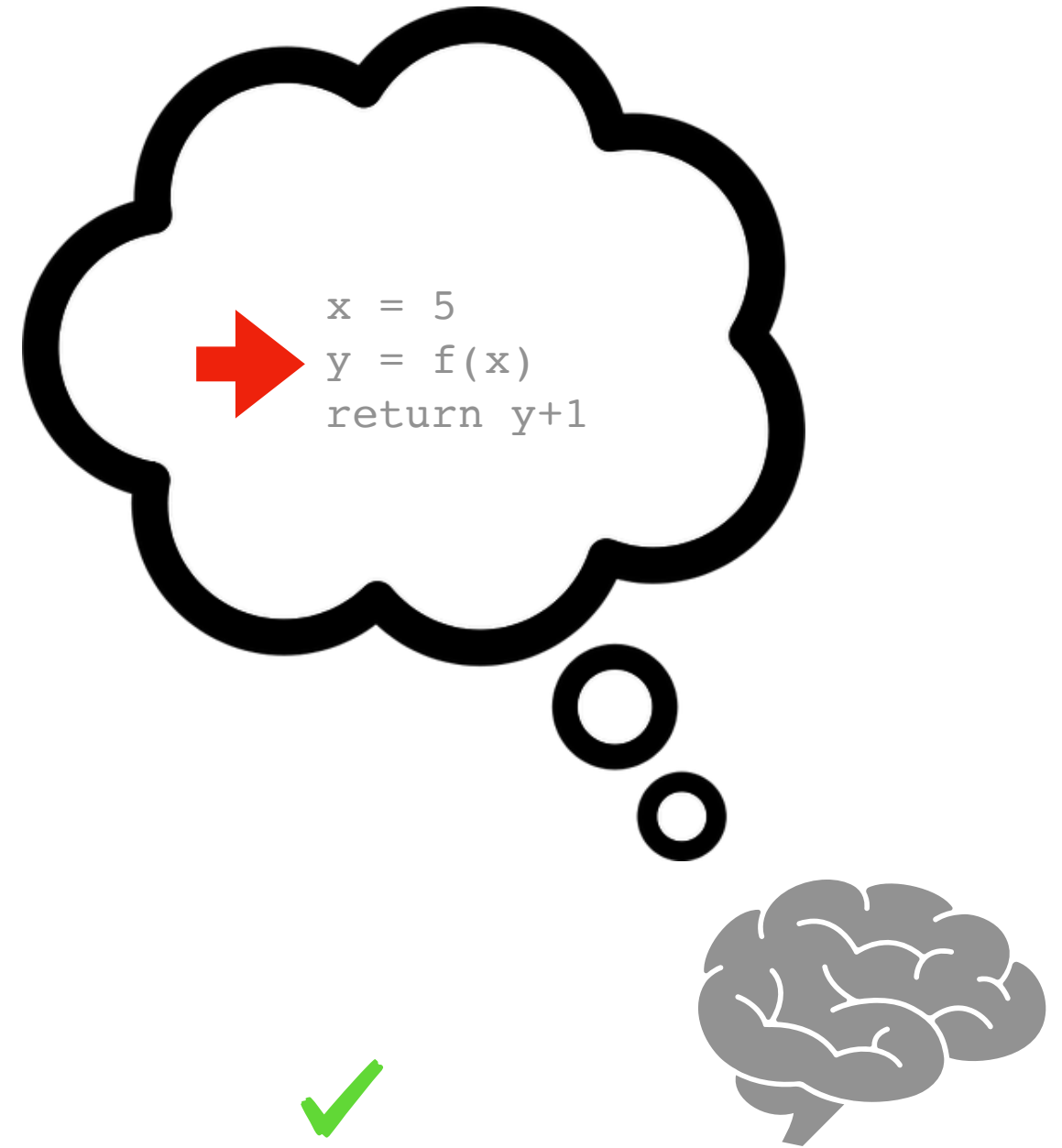
Due: Quiz2

Part I: Intro to conditionals

Mental Model of Control Flow

Code:

```
...  
x = 5  
y = f(x)  
return y+1  
...
```



three
exceptions

1. do statements in order, one at a time
2. functions: jump in and out of these
3. conditionals: sometimes skip statements
4. loops: sometimes go back to previous



← TODAY

Learning Objectives Today

Write conditional statements

- Conditional execution (if)
- Alternate execution (else)
- Chained conditionals (elif)

Chapter 5 of Think Python
(skip "Recursion" sections)

Do PythonTutor Practice!
(posted on schedule)

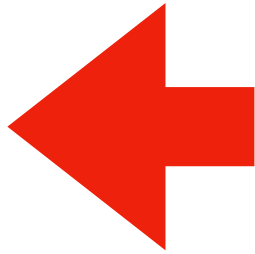
Determine the output of conditional statements

Identify nested code blocks

- Count the number of blocks in a segment of code

Today's Outline

Review



Control Flow Diagrams

Basic syntax for “if”

Demos

Identifying code blocks

Demos / worksheet

Review I: Indentation Example

what does it print?

```
print("A")
```

```
print("B")
```

```
def print_letters():
```

```
    print("C")
```

```
    print("D")
```

```
print("E")
```

```
print("F")
```

```
print_letters()
```

Review I: Indentation Example

what does it print?

```
print("A")
```

```
print("B")
```

```
def print_letters():
```

```
    print("C")
```

```
    print("D")
```

```
print("E")
```

```
print("F")
```

```
print_letters()
```

A
B
E
F
C
D

Review I: Indentation Example

what does it print?

```
print("A")
```

```
print("B")
```

```
def print_letters():
```

```
    print("C")
```

```
    print("D")
```

indented, so "inside"
print_letters function

```
print("E")
```

```
print("F")
```

```
print_letters()
```

A
B
E
F
C
D

Review I: Indentation Example

what does it print?

```
print("A")
```

```
print("B")
```

```
def print_letters():
```

```
    print("C")
```

```
    print("D")
```

indented, so "inside"
print_letters function

```
print("E")
```

```
print("F")
```

printed last because
print_letters is called last

```
print_letters()
```

A

B

E

F

C

D

Review I: Indentation Example

```
print("A")  
print("B")
```

not indented, so
“outside” any function

```
def print_letters():
```

```
    print("C")  
    print("D")
```

indented, so “inside”
print_letters function

```
print("E")
```

```
print("F")
```

```
print_letters()
```

what does it print?

A
B
E
F
C
D

Review I: Indentation Example

what does it print?

```
print("A")  
print("B")
```

not indented, so
“outside” any function

```
def print_letters():
```

```
    print("C")  
    print("D")
```

indented, so “inside”
print_letters function

```
print("E")  
print("F")
```

also not indented, so
“outside” any function.
Runs BEFORE
print_letters is called

```
print_letters()
```

A
B
E
F
C
D

Review I: Indentation Example

what does it print?

```
print("A")  
print("B")
```

not indented, so
“outside” any function

```
def print_letters():
```

```
    print("C")  
    print("D")
```

indented, so “inside”
print_letters function

```
print("E")  
print("F")
```

also not indented, so
“outside” any function.
Runs BEFORE
print_letters is called

blank lines are irrelevant

```
print_letters()
```

A
B
E
F
C
D

We use indenting to tell Python which code is inside or outside of a function (or other things we'll learn about soon).

Review I: Indentation Example

what does it print?

```
print("A")
```

```
print("B")
```

```
def print_letters():
```

```
    print("C")
```

```
    print("D")
```

we'll often call the lines
of code inside something
a "block" of code

```
print("E")
```

```
print("F")
```

```
print_letters()
```

A
B
E
F
C
D

Review I: Indentation Example

what does it print?

```
print("A")
```

```
print("B")
```

```
def print_letters():
```

```
    print("C")
```

```
    print("D")
```

```
print("E")
```

```
print("F")
```

```
print_letters()
```

horizontal spaces
identify blocks (not
vertical space)

A
B
E
F
C
D

Review 2: Argument Passing

```
def h(x=1, y=2):  
    print(x, y)    # what is printed?
```

```
def g(x, y):  
    print(x, y)    # what is printed?  
    h(y)
```

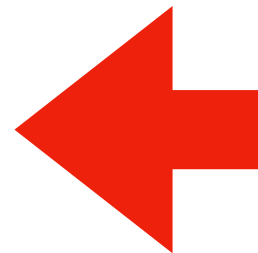
```
def f(x, y):  
    print(x, y)    # what is printed?  
    g(x=x, y=y+1)
```

```
x = 10  
y = 20  
f(y, x)
```

Today's Outline

Review

Control Flow Diagrams



Basic syntax for “if”

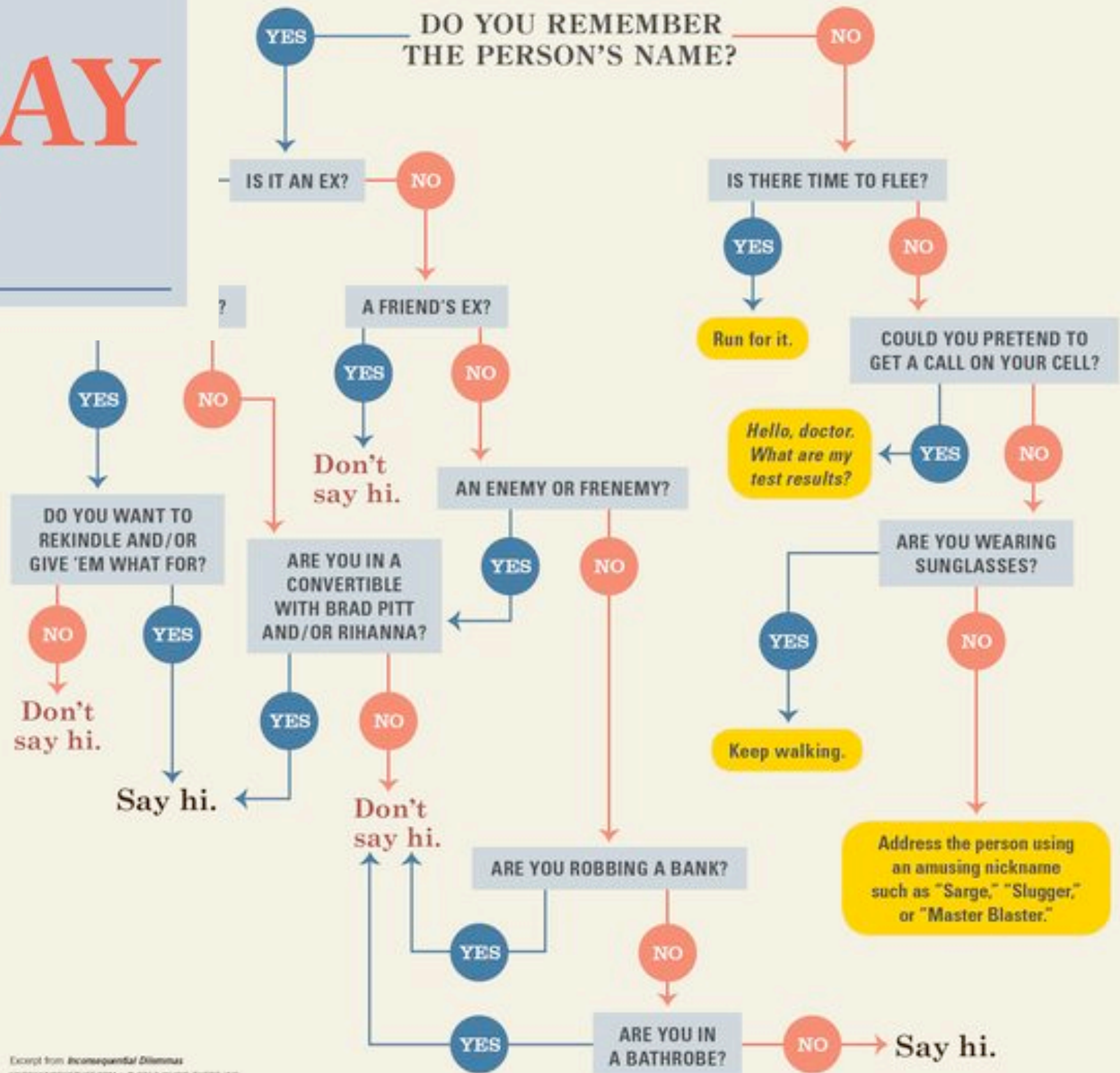
Demos

Identifying code blocks

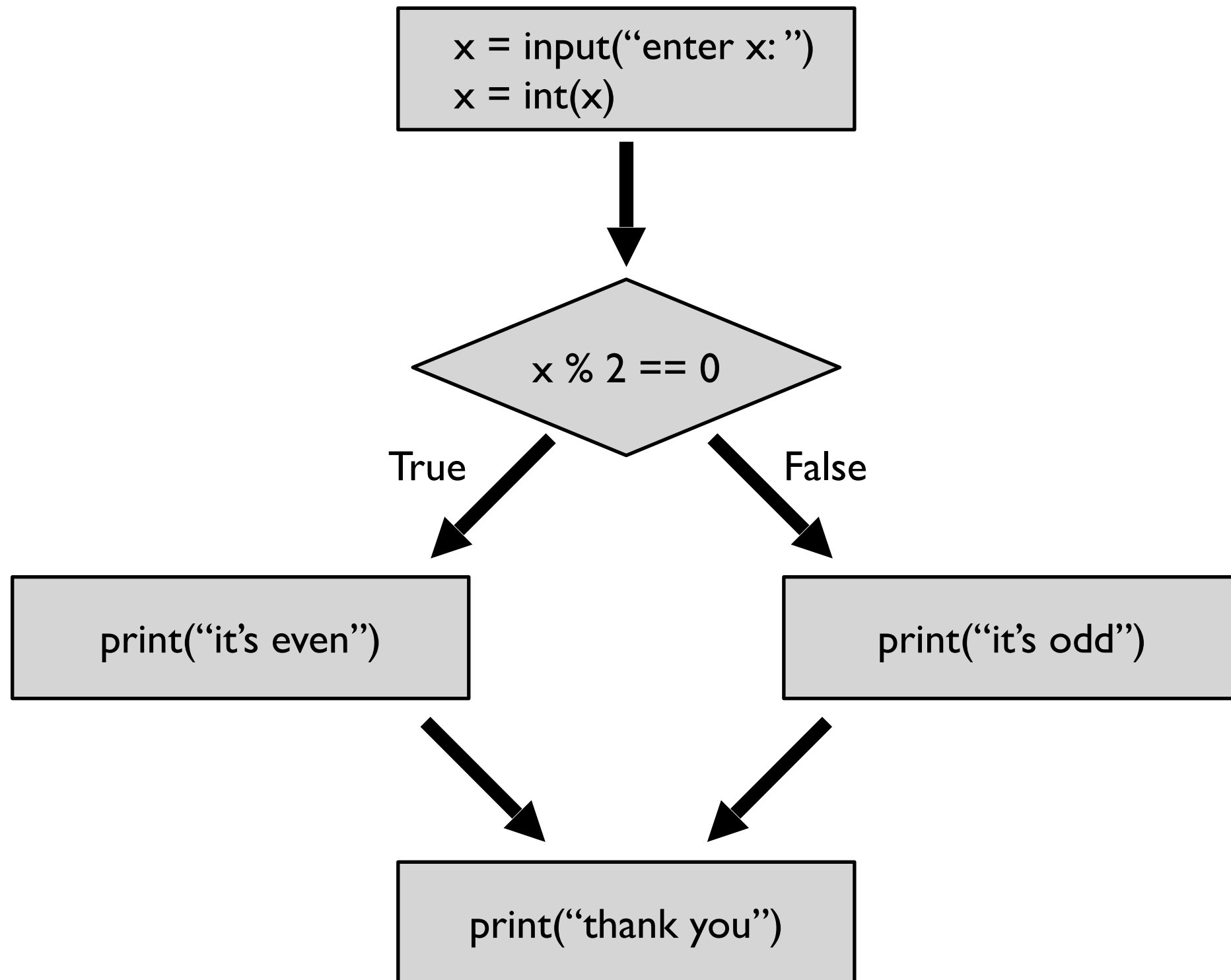
Demos / worksheet

*I just
saw someone
I know.*

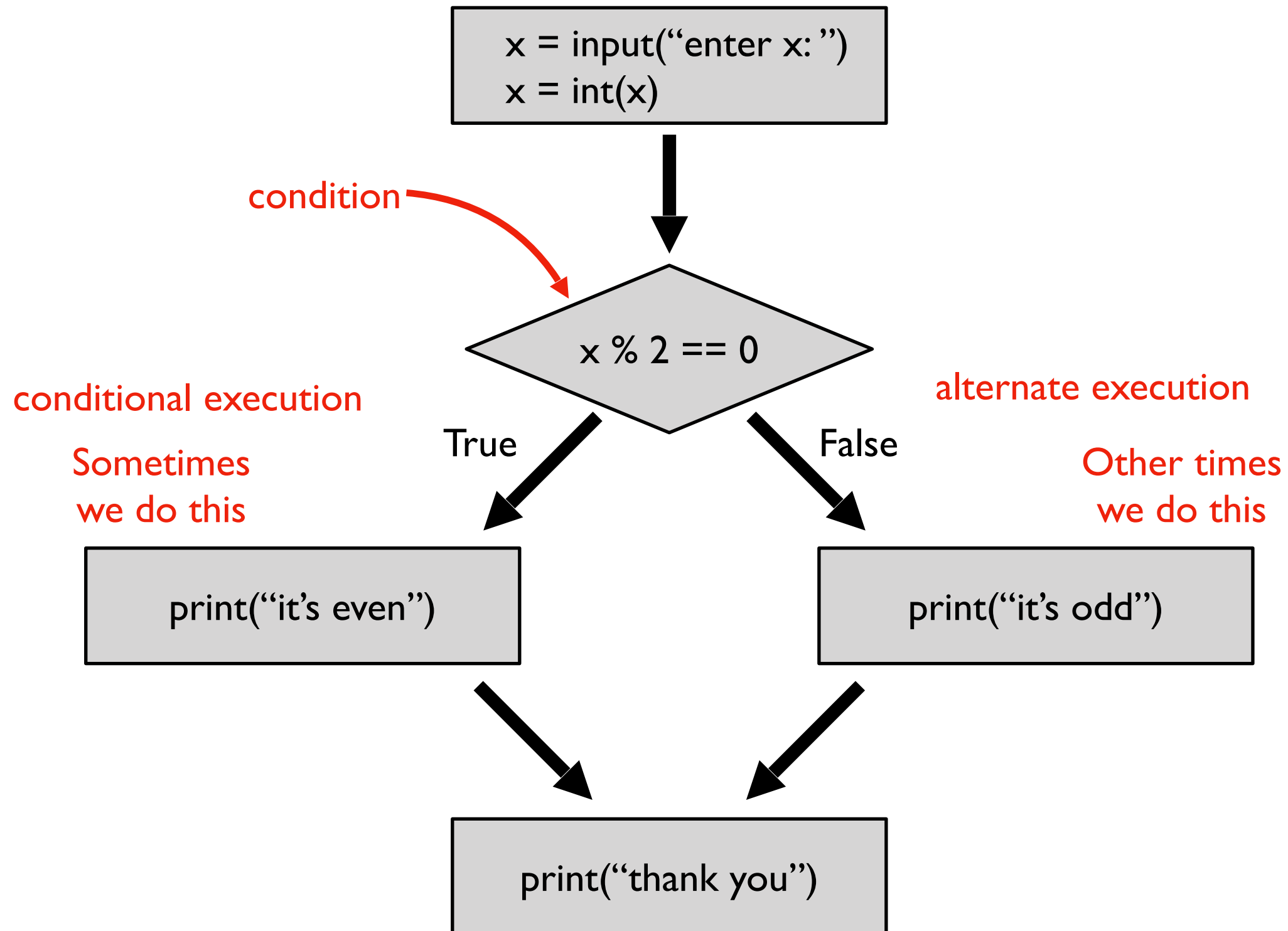
DO I SAY HI?



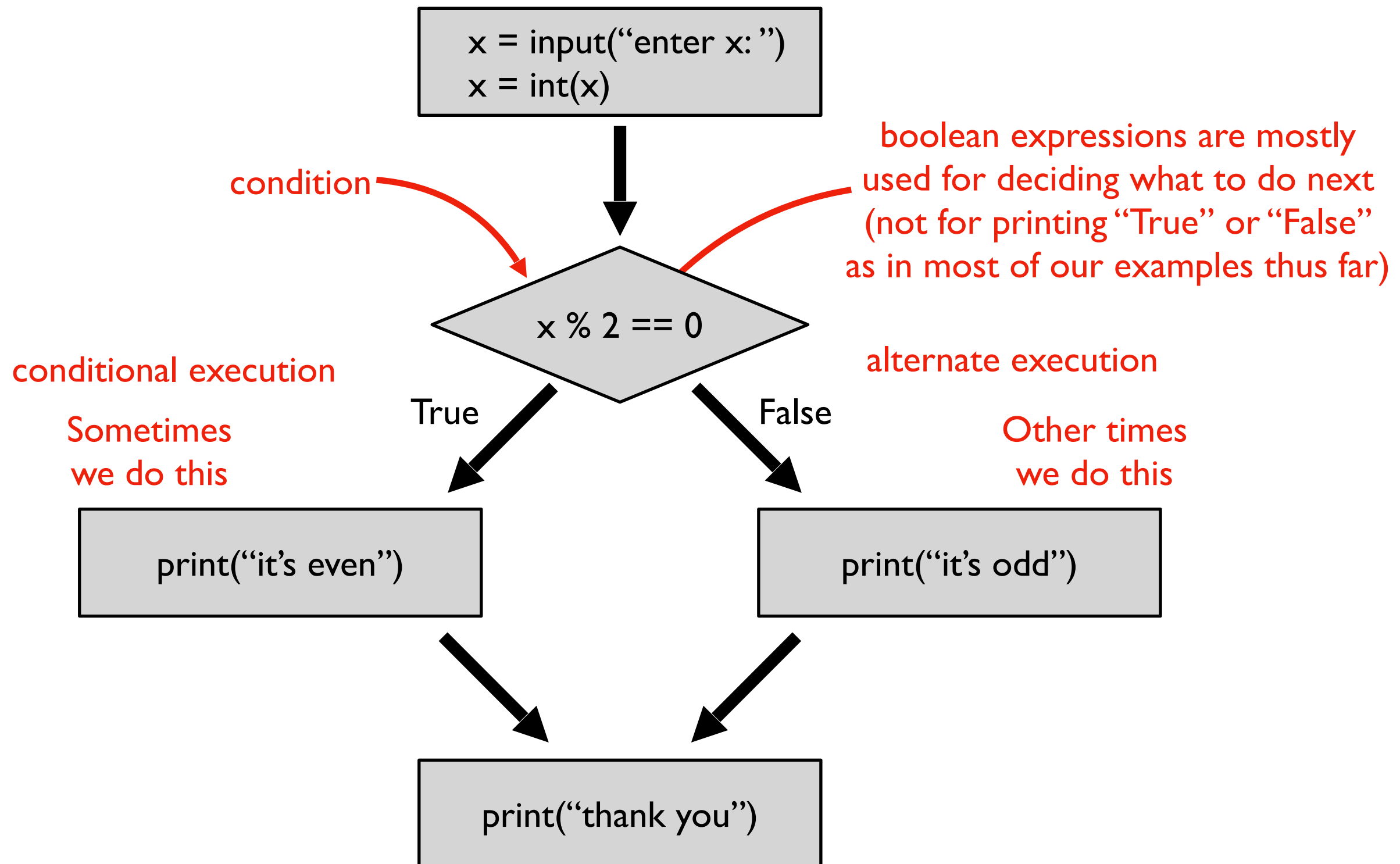
Control Flow Diagrams (Flowcharts for Code)



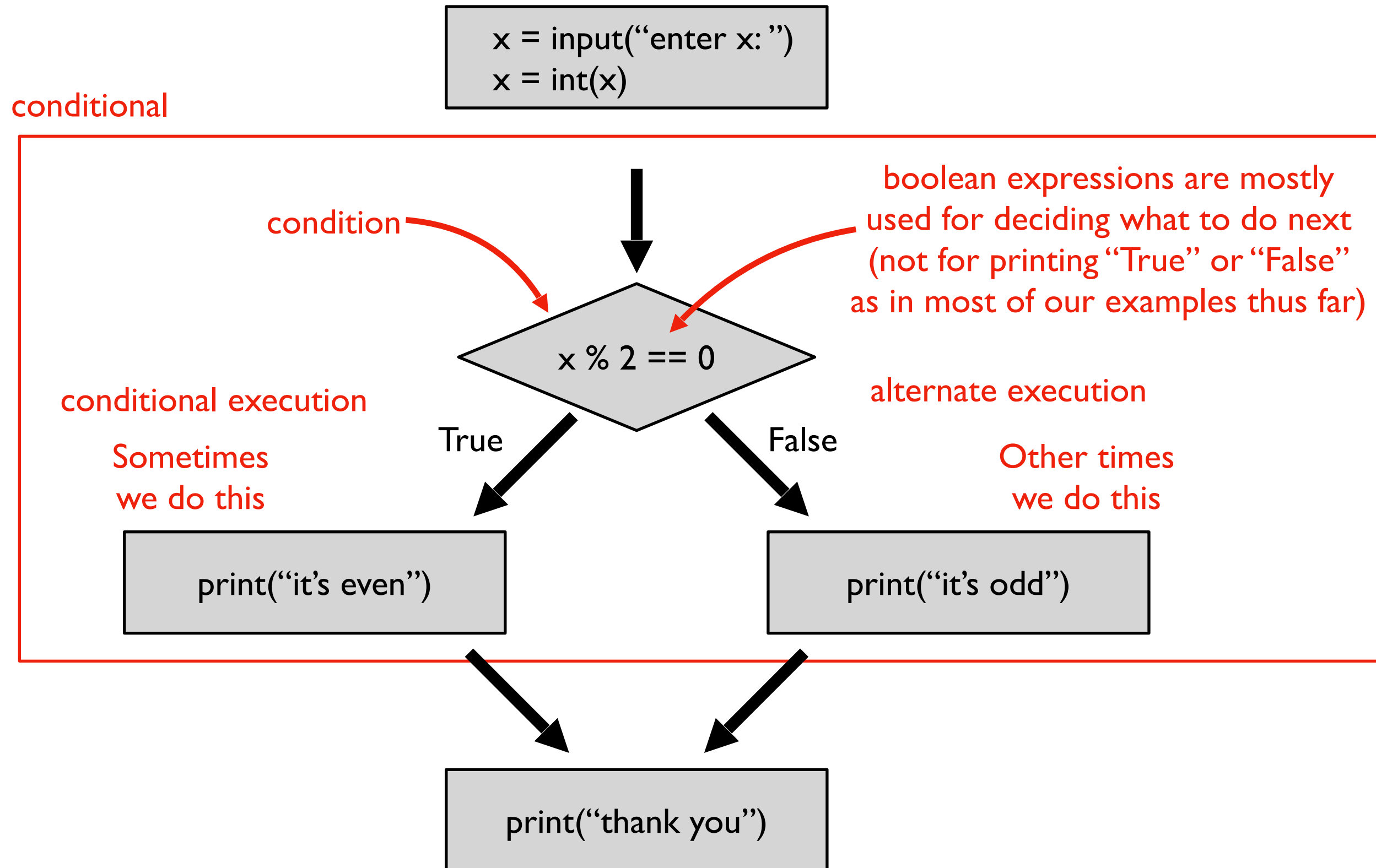
Control Flow Diagrams (Flowcharts for Code)



Control Flow Diagrams (Flowcharts for Code)



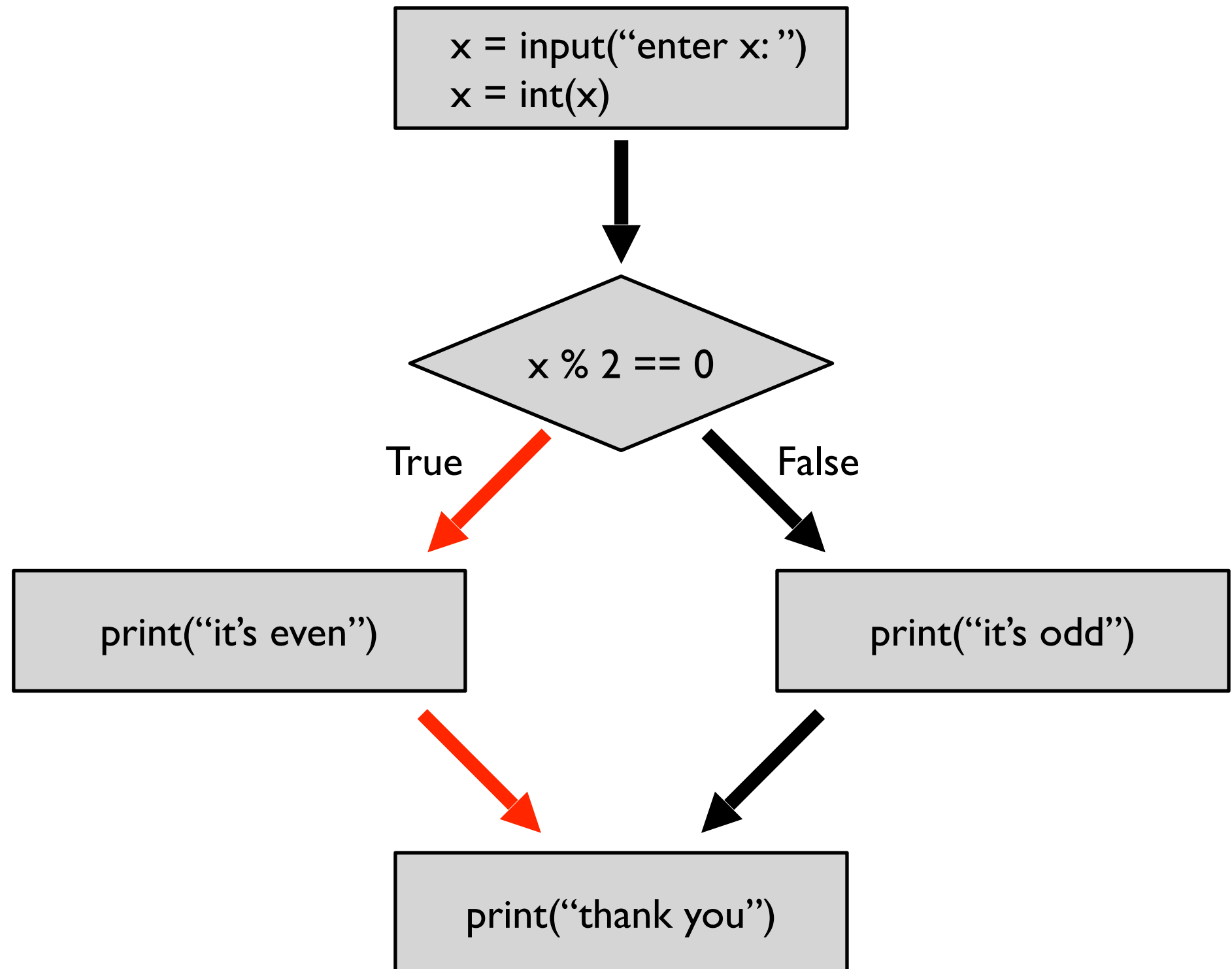
Control Flow Diagrams (Flowcharts for Code)



Branches (aka "Paths of Execution")

Input/Output:

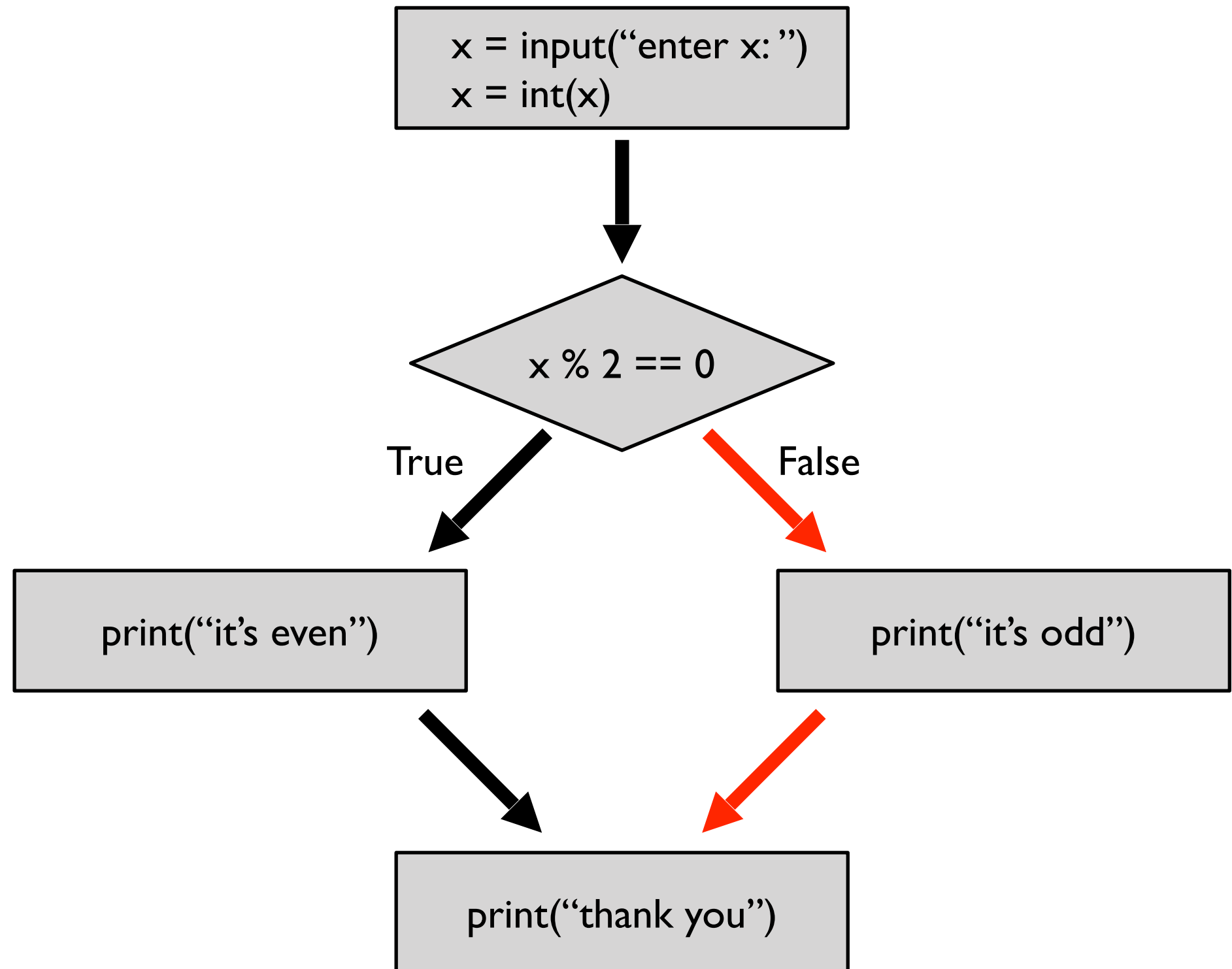
```
enter x: 8  
it's even  
thank you
```



Branches (aka "Paths of Execution")

Input/Output:

```
enter x: 7  
it's odd  
thank you
```

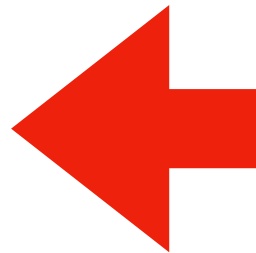


Today's Outline

Review

Control Flow Diagrams

Basic syntax for “if”



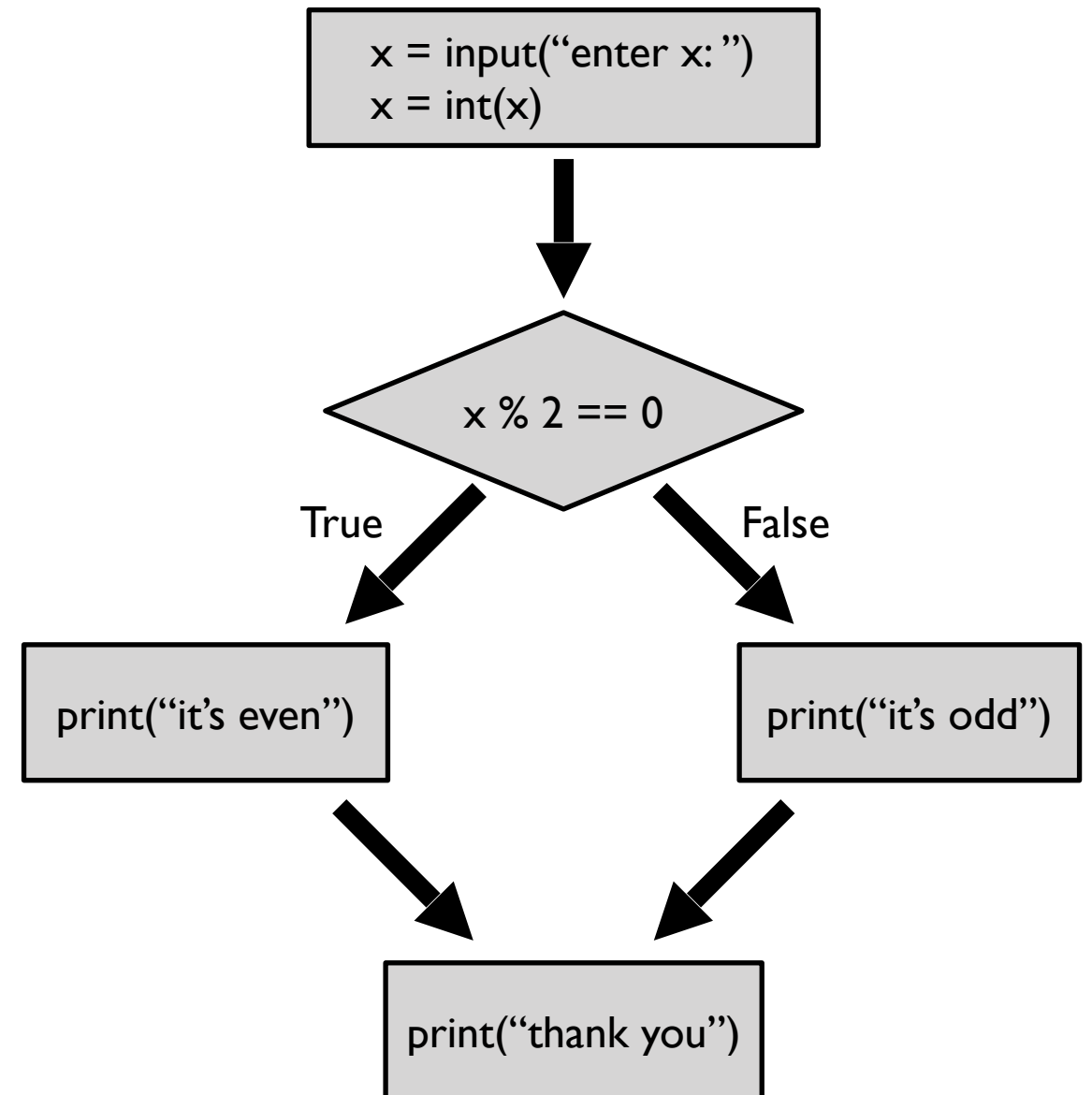
Demos

Identifying code blocks

Demos / worksheet

Writing conditions in Python

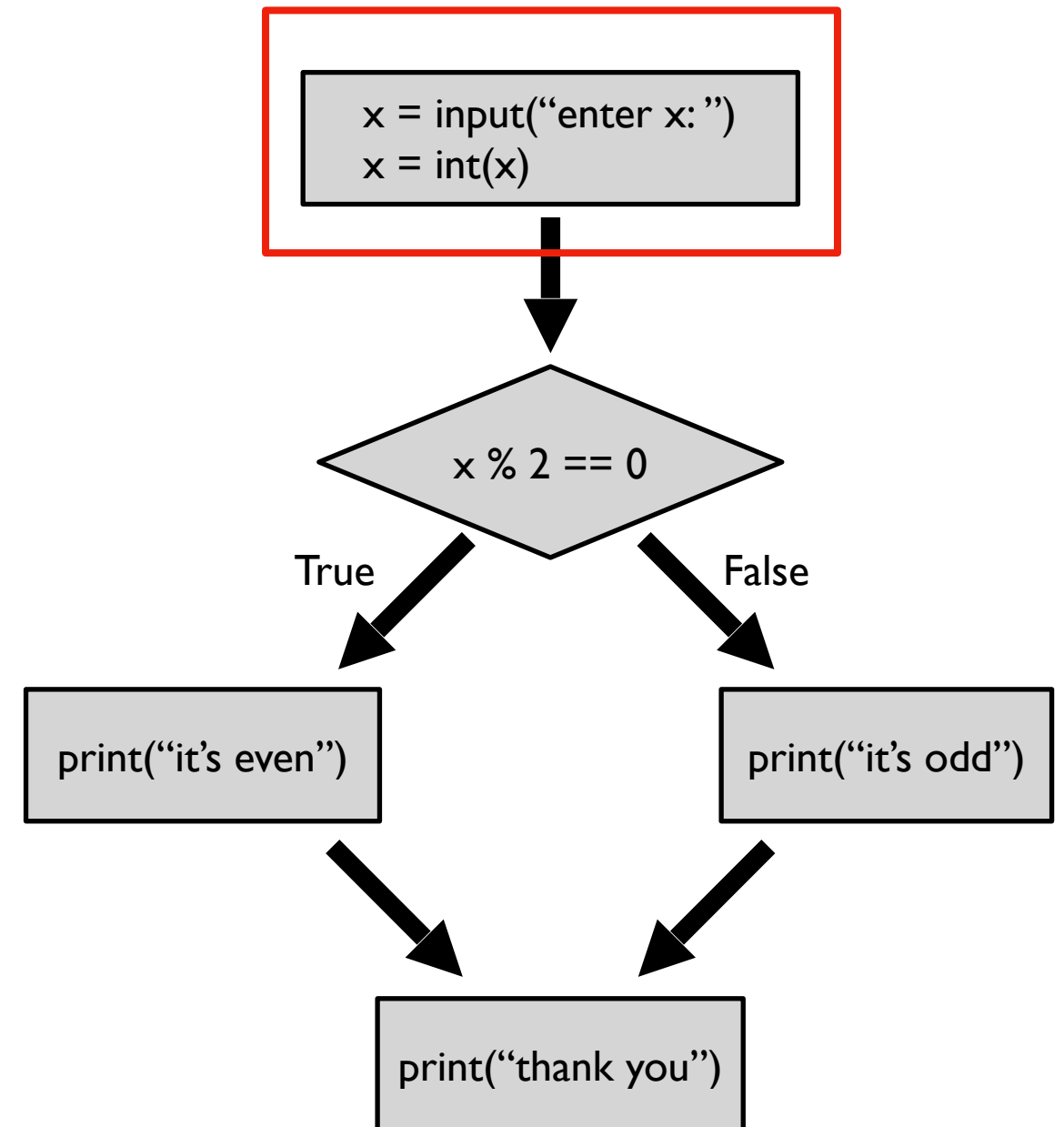
Code:



Writing conditions in Python

Code:

```
x = input("enter x: ")  
x = int(x)
```

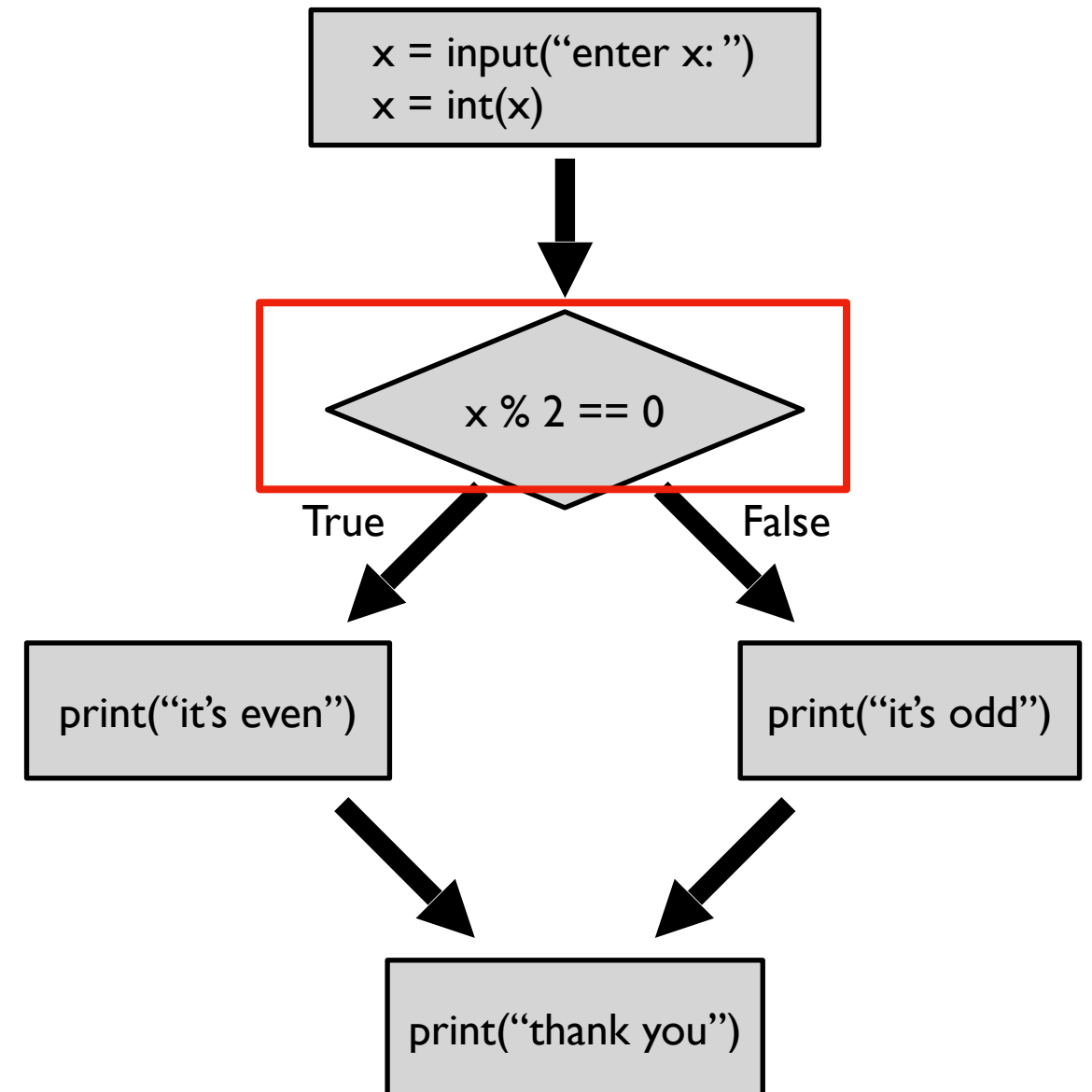


Writing conditions in Python

Code:

```
x = input("enter x: ")  
x = int(x)
```

```
if x % 2 == 0:
```

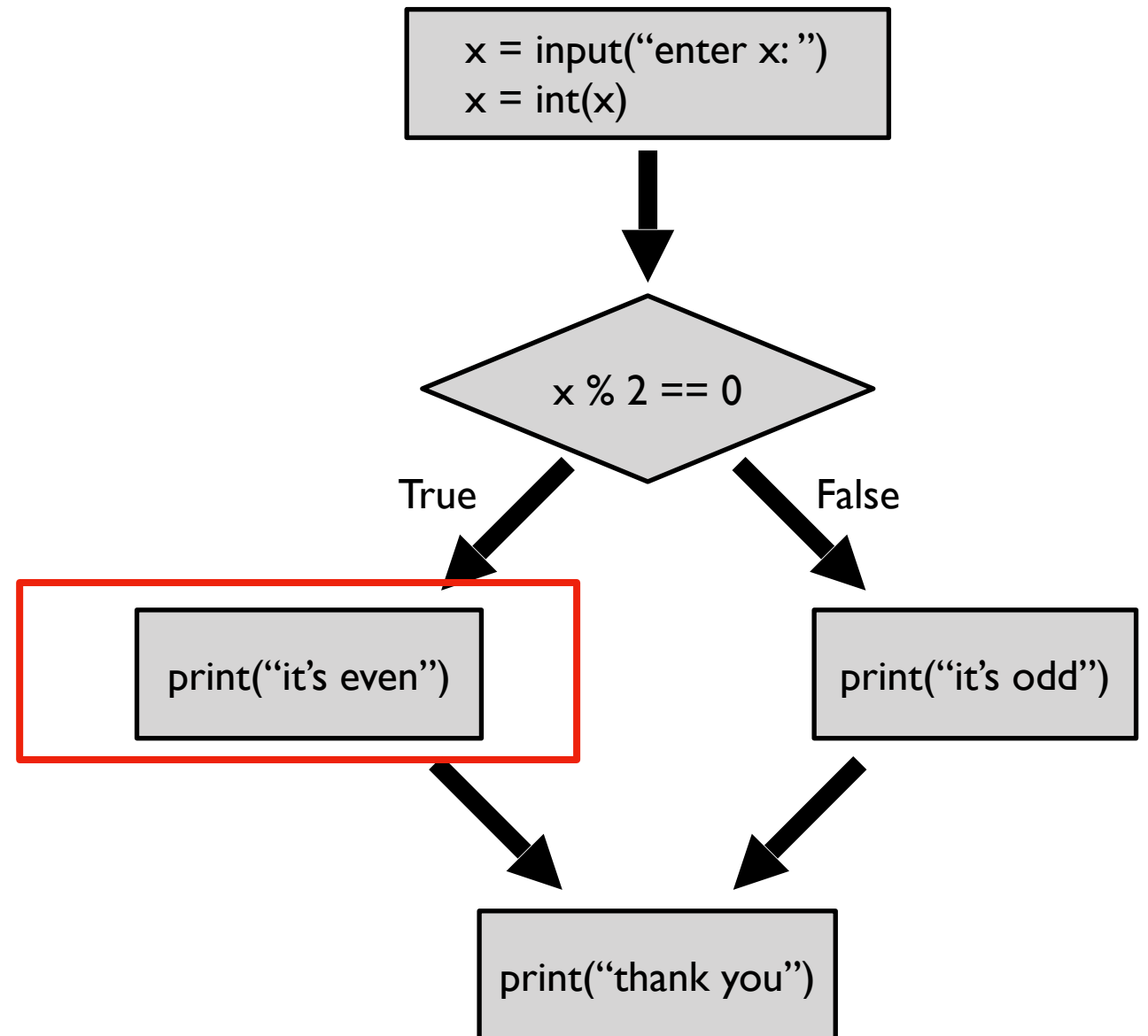


Writing conditions in Python

Code:

```
x = input("enter x: ")  
x = int(x)
```

```
if x % 2 == 0:  
    print("it's even")
```



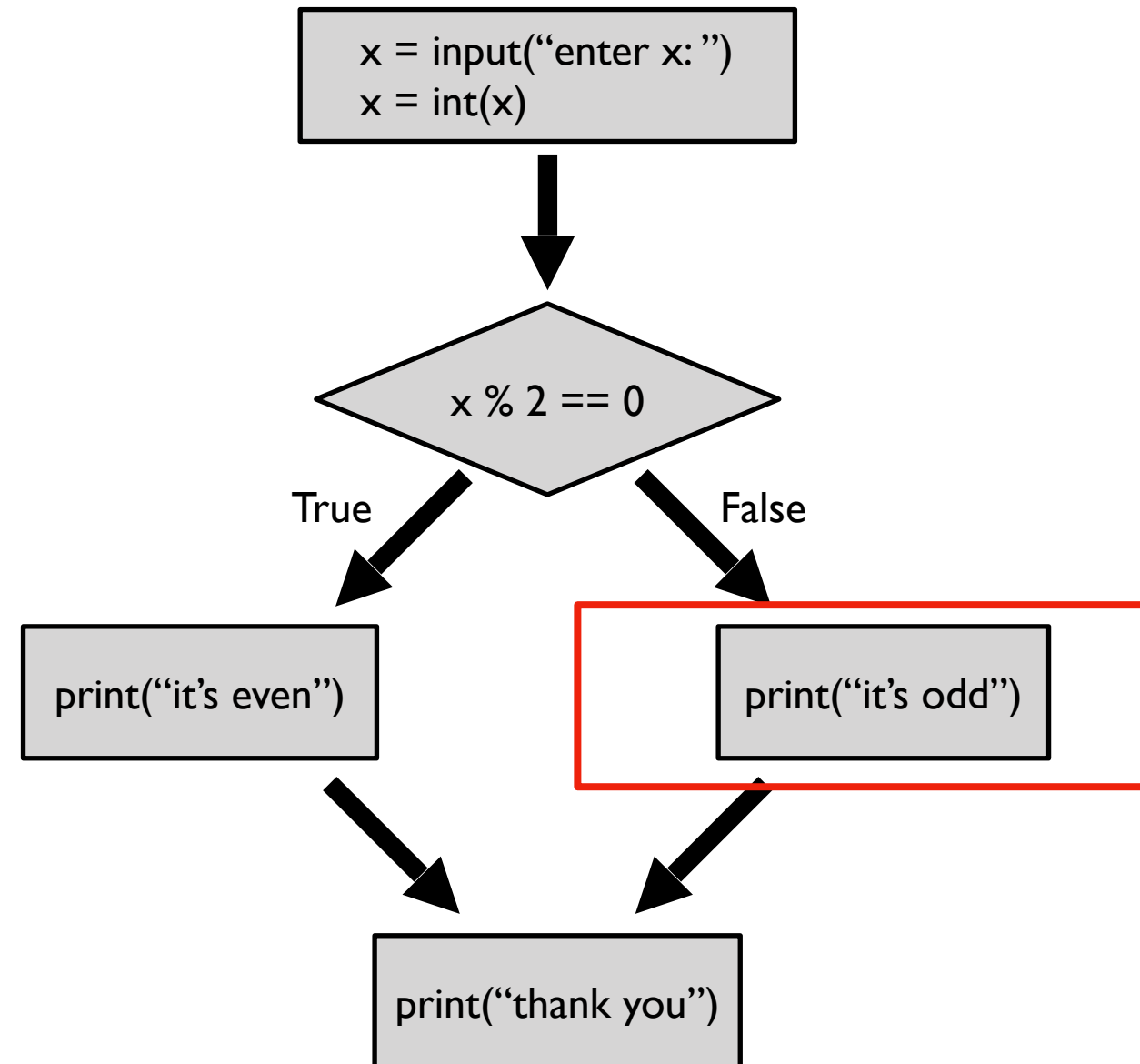
Writing conditions in Python

Code:

```
x = input("enter x: ")  
x = int(x)
```

```
if x % 2 == 0:  
    print("it's even")  
else:  
    print("it's odd")
```

colons will *almost* always be followed by a tabbed new line



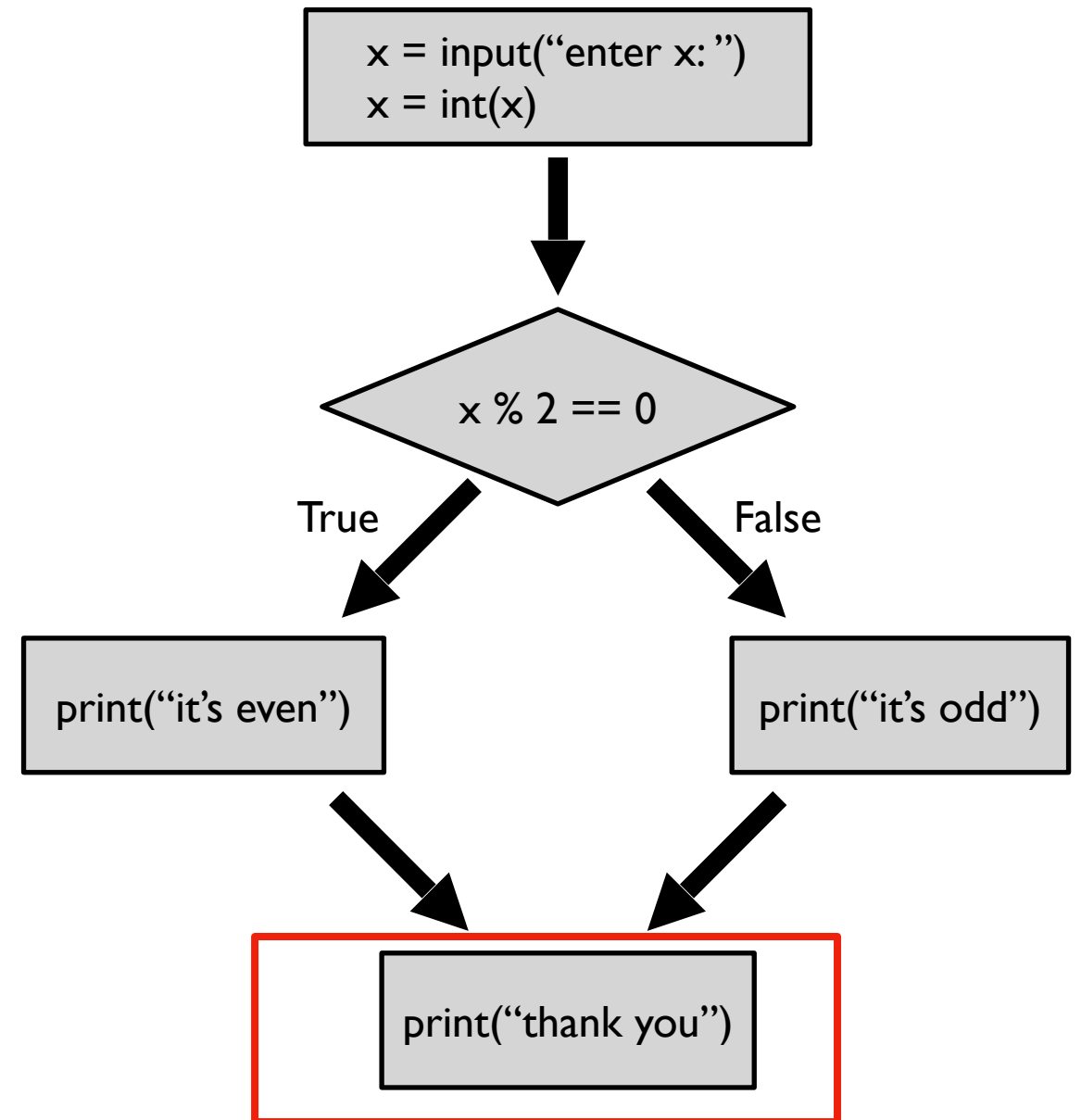
Writing conditions in Python

Code:

```
x = input("enter x: ")  
x = int(x)
```

```
if x % 2 == 0:  
    print("it's even")  
else:  
    print("it's odd")
```

```
print("thank you")
```



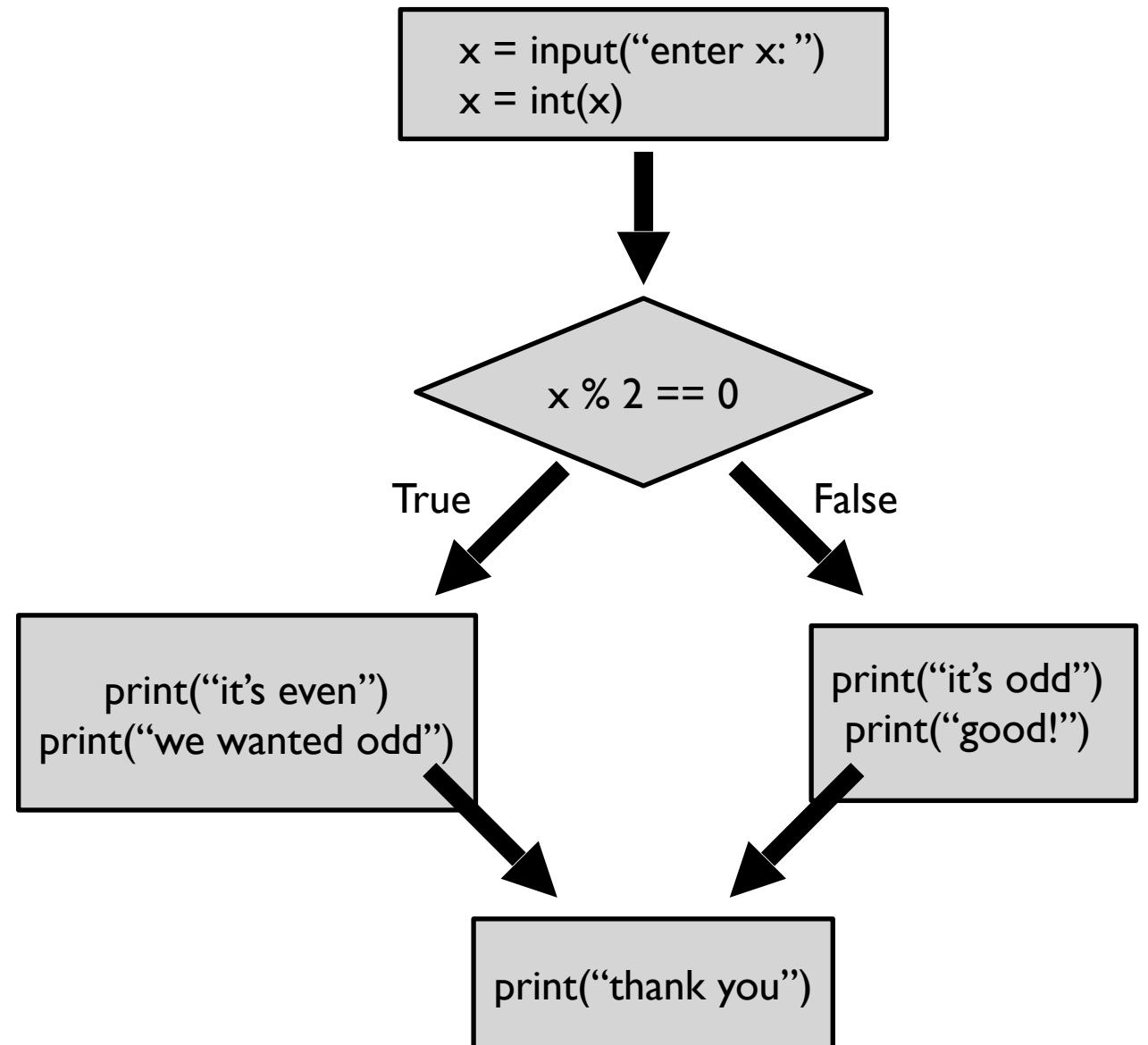
Writing conditions in Python

Code:

```
x = input("enter x: ")
x = int(x)

if x % 2 == 0:
    print("it's even")
    print("we wanted odd")
else:
    print("it's odd")
    print("good!")

print("thank you")
```



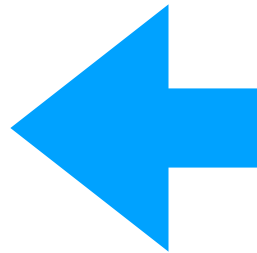
Today's Outline

Review

Control Flow Diagrams

Basic syntax for “if”

Demos



Identifying code blocks

Demos / worksheet

Today's Outline

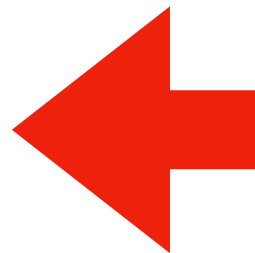
Review

Control Flow Diagrams

Basic syntax for “if”

Demos

Identifying code blocks



Demos / worksheet

Code Blocks

Code:

```
x = input("enter x: ")  
x = int(x)
```

```
if x % 2 == 0:
```

```
    print("it's even")  
    print("we wanted odd")
```

block of code
inside "if"

```
else:
```

```
    print("it's odd")  
    print("good!")
```

block of code
inside "else"

```
print("thank you")  
print("all done")
```

Code Blocks

Code:

```
x = input("enter x: ")  
x = int(x)
```

```
if x % 2 == 0:
```

```
    print("it's even")  
    print("we wanted odd")
```

block of code
inside "if"

```
else:
```

```
    print("it's odd")  
    print("good!")
```

block of code
inside "else"

```
print("thank you")  
print("all done")
```

What if all this were inside a function?

Code Blocks

You need to get good at “seeing” code blocks in Python code.
Even blocks inside blocks inside blocks...

Code:

```
def check_oddness():
```

```
    x = input("enter x: ")
```

```
    x = int(x)
```

```
    if x % 2 == 0:
```

```
        print("it's even")
```

```
        print("we wanted odd")
```

```
    else:
```

```
        print("it's odd")
```

```
        print("good!")
```

```
    print("thank you")
```

```
    print("all done")
```

```
check_oddness()
```

block of code
inside “if”

block of code
inside “else”

block of code in
check_oddness

Identifying Code Blocks

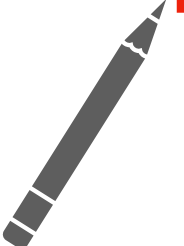
Code:

```
def check_oddness():  
    x = input("enter x: ")  
    x = int(x)  
  
    if x % 2 == 0:  
        print("it's even")  
        print("we wanted odd")  
    else:  
        print("it's odd")  
        print("good!")  
  
    print("thank you")  
    print("all done")  
  
check_oddness()
```

Step 1: look for a colon at
end of a line

Identifying Code Blocks

Code:



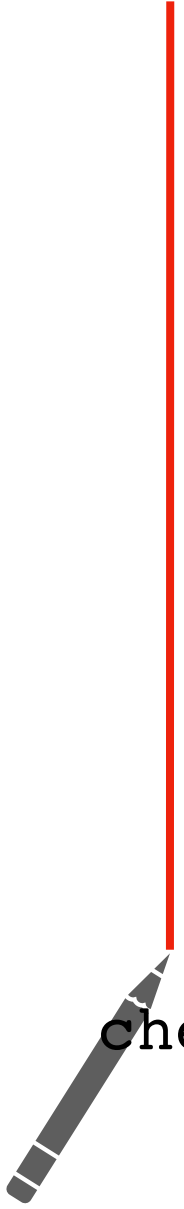
```
def check_oddness():  
    x = input("enter x: ")  
    x = int(x)  
  
    if x % 2 == 0:  
        print("it's even")  
        print("we wanted odd")  
    else:  
        print("it's odd")  
        print("good!")  
  
    print("thank you")  
    print("all done")  
  
check_oddness()
```

Step 2: start drawing a line
on next code line, indented in

Identifying Code Blocks

Code:

```
def check_oddness():  
    x = input("enter x: ")  
    x = int(x)  
  
    if x % 2 == 0:  
        print("it's even")  
        print("we wanted odd")  
    else:  
        print("it's odd")  
        print("good!")  
  
    print("thank you")  
    print("all done")  
  
check_oddness()
```



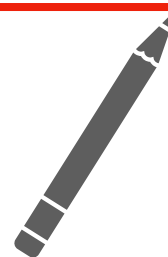
Step 3: continue down until you hit
code that is less indented

Identifying Code Blocks

Code:

```
def check_oddness():  
    x = input("enter x: ")  
    x = int(x)  
  
    if x % 2 == 0:  
        print("it's even")  
        print("we wanted odd")  
    else:  
        print("it's odd")  
        print("good!")  
  
    print("thank you")  
    print("all done")  
  
check_oddness()
```

Step 4: box off the code



Identifying Code Blocks

Code:

```
def check_oddness():  
    x = input("enter x: ")  
    x = int(x)  
  
    if x % 2 == 0:  
        print("it's even")  
        print("we wanted odd")  
    else:  
        print("it's odd")  
        print("good!")  
  
    print("thank you")  
    print("all done")
```

```
check_oddness()
```

Step 4: box off the code

Identifying Code Blocks

Code:

```
def check_oddness():  
    x = input("enter x: ")  
    x = int(x)  
  
    if x % 2 == 0:  
        print("it's even")  
        print("we wanted odd")  
    else:  
        print("it's odd")  
        print("good!")  
  
    print("thank you")  
    print("all done")
```

check_oddness()

to find more boxes,
look for the next colon
and repeat

Identifying Code Blocks

Code:

```
def check_oddness():
```

```
    x = input("enter x: ")
```

```
    x = int(x)
```

```
    if x % 2 == 0:
```

```
        print("it's even")
```

```
        print("we wanted odd")
```

```
    else:
```

```
        print("it's odd")
```

```
        print("good!")
```

```
    print("thank you")
```

```
    print("all done")
```

```
check_oddness()
```

to find more boxes,
look for the next colon
and repeat

Identifying Code Blocks

Code:

```
def check_oddness():
```

```
    x = input("enter x: ")
```

```
    x = int(x)
```

```
    if x % 2 == 0:
```

```
        print("it's even")
```

```
        print("we wanted odd")
```

```
    else:
```

```
        print("it's odd")
```

```
        print("good!")
```

```
    print("thank you")
```

```
    print("all done")
```

```
check_oddness()
```

to find more boxes,
look for the next colon
and repeat

Identifying Code Blocks

Code:

```
def check_oddness():
```

```
    x = input("enter x: ")  
    x = int(x)
```

```
    if x % 2 == 0:
```

```
        print("it's even")  
        print("we wanted odd")
```

```
    else:
```

```
        print("it's odd")  
        print("good!")
```

```
    print("thank you")  
    print("all done")
```

```
check_oddness()
```

to find more boxes,
look for the next colon
and repeat

Today's Outline

Review

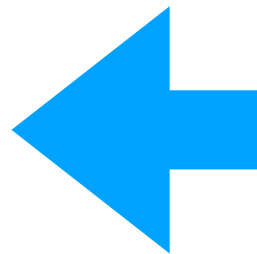
Control Flow Diagrams

Basic syntax for “if”

Demos

Identifying code blocks

Demos / worksheet



Practice Problems

Example: Classifying Children by Age

What are all the different ways to classify children?

If you are 3 years old you are a

If you are 15 years old you are a

Write a function that is given an int and returns a string

```
def categorize_age(age):  
    if age <= ....  
        return 'baby'
```


Example: Date Printer

```
please enter a year: (YYYY): 2022
please enter a month (1-12): 2
please enter a day (1-31): 11
the date is: Feb 11th of '22
```

convert month num to name

'2-digit year

e.g., 1st, 2nd, 3rd, etc



Part 2: nesting and refactoring conditionals

Learning Objectives

Write nested conditional statements

Refactor code with Boolean operators into equivalent code with nested conditional statements

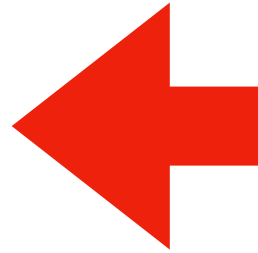
Refactor code with nested conditional statements into equivalent code with Boolean operators

Identify code blocks

- Count the number of blocks in nested code

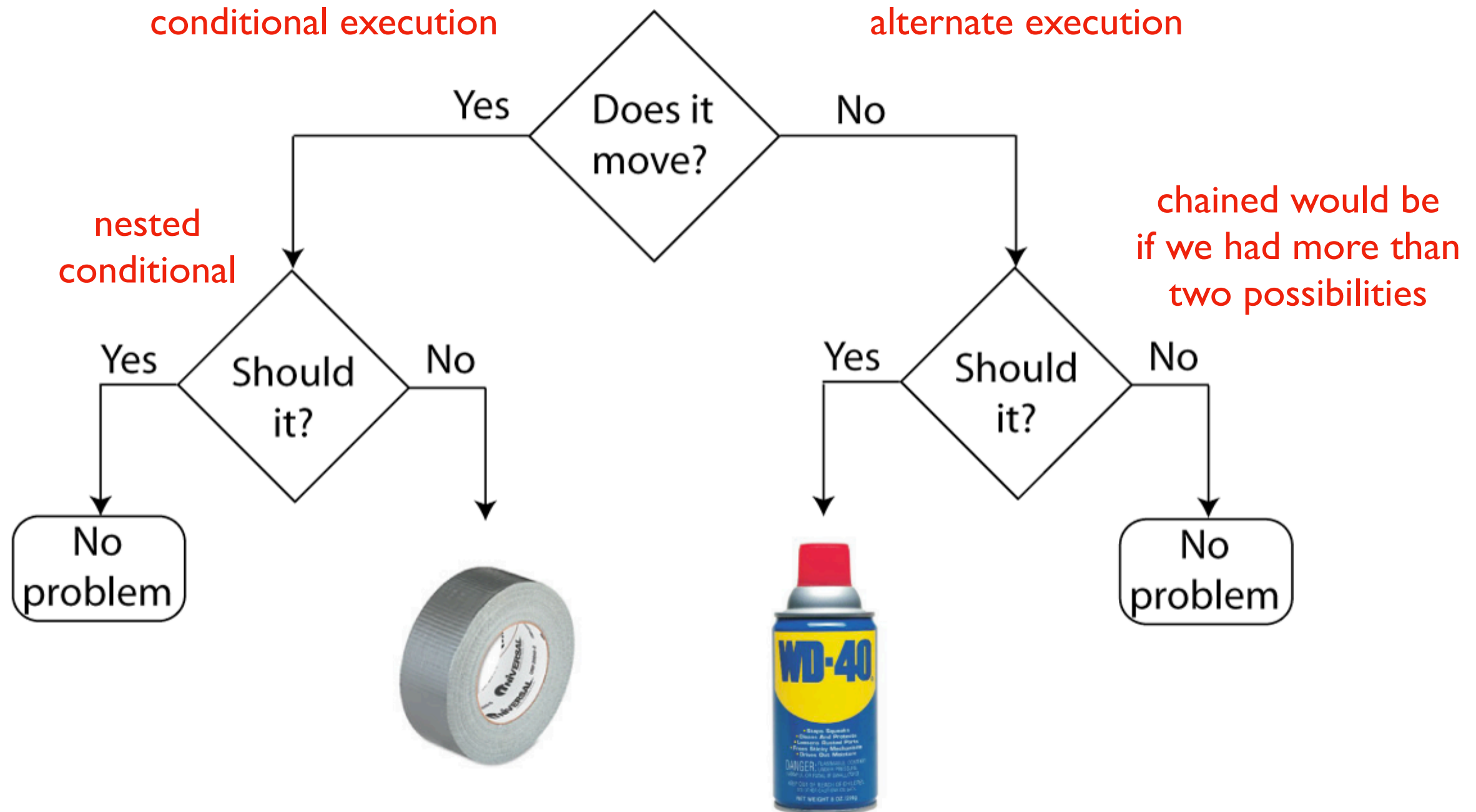
Today's Outline

Nested Conditionals



Refactoring Conditionals

Laboratory Troubleshooting Flowchart



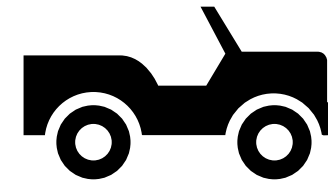
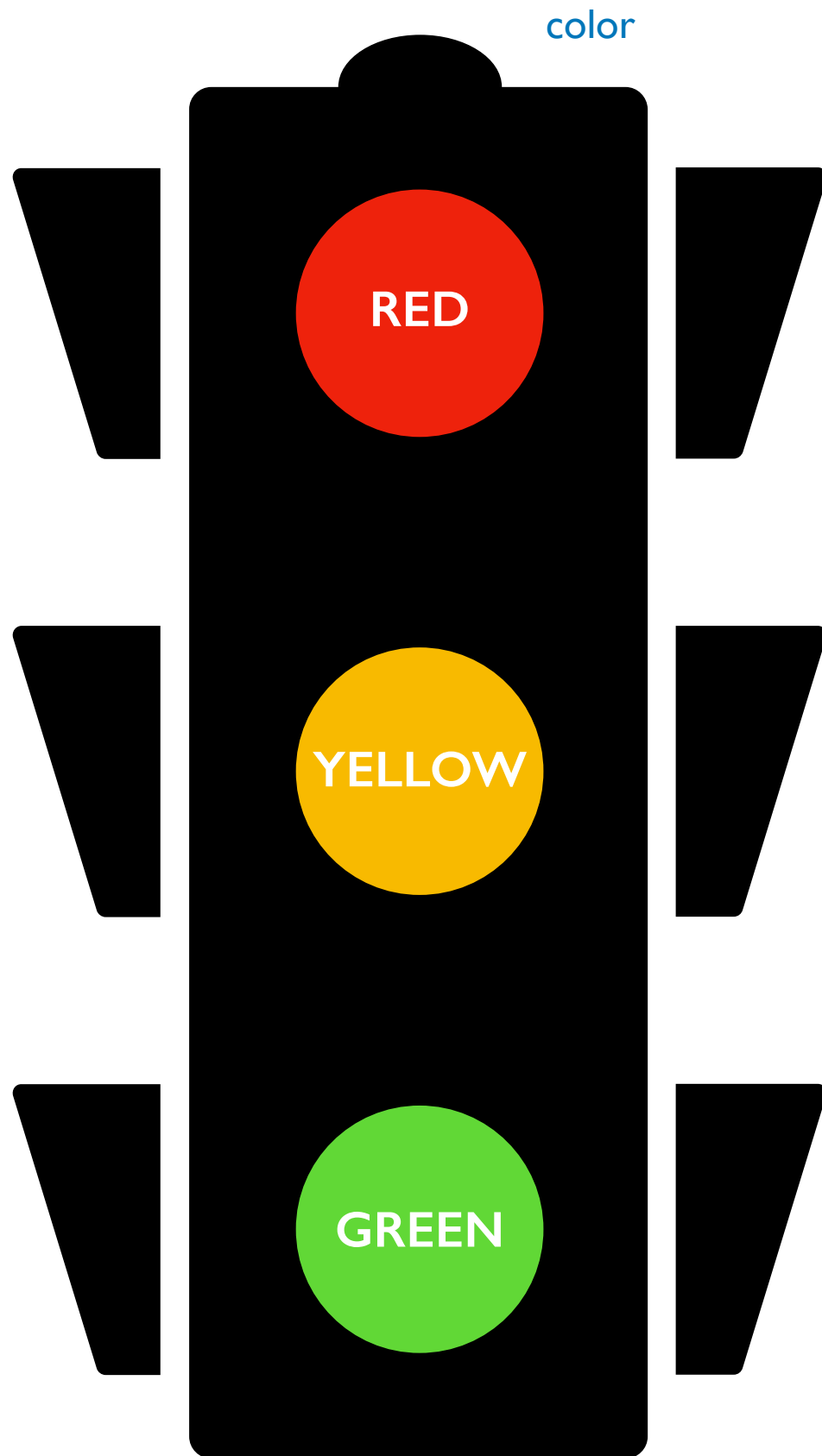
in programming:

- **questions** are phrased as *boolean expressions*
- **actions** are *code/statements*

Nested Conditionals Example

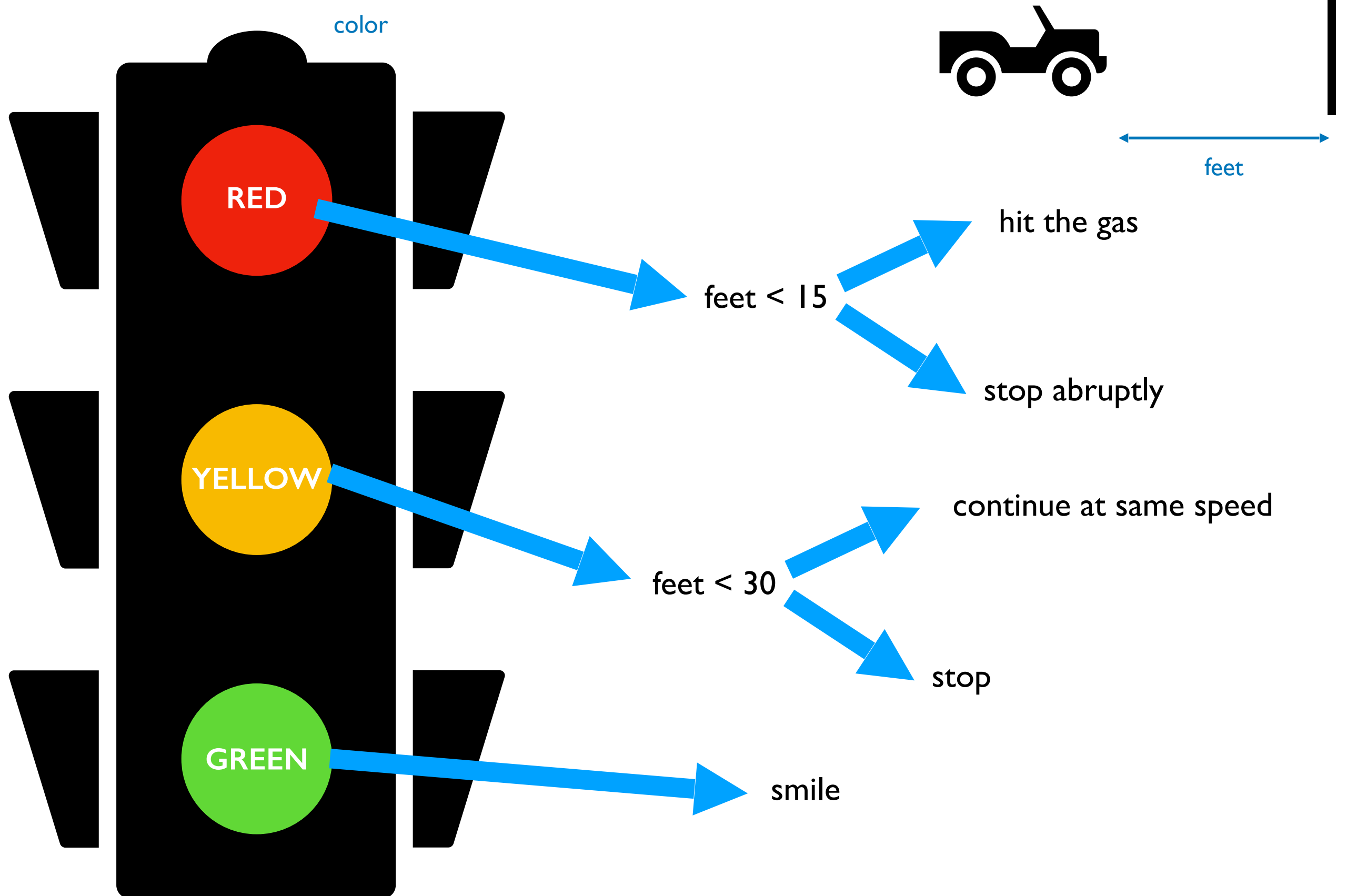
```
def fix(moves, should):  
    if moves:  
        if should:  
            return "good"  
        else:  
            return "duct tape"  
    else:  
        if should:  
            return "WD-40"  
        else:  
            return "good"
```

Example: Stoplight



what should the driver do?

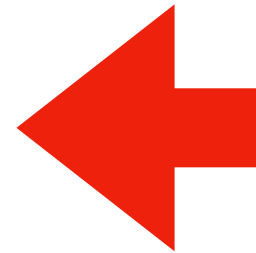
Example: Stoplight



Today's Outline

Nested Conditionals

Refactoring Conditionals



How to use these slides

There are more examples here than we can cover in lecture.

However, you can walk through these examples along with the interactive exercises. You should do the following:


1. Think about what the answer is
2. Mentally step through the code using the example call when applicable
3. Step through the code with the Python Tutor examples we've setup for you. For the refactor examples, step through all three versions, and see which alternative (A or B) matches the output of the original version.
4. If you got something different than Python Tutor, tweak your mental model (talk to us if you don't understand something)

Refactor Exercise 1

```
def or2(cond1, cond2):  
    return cond1 or cond2
```

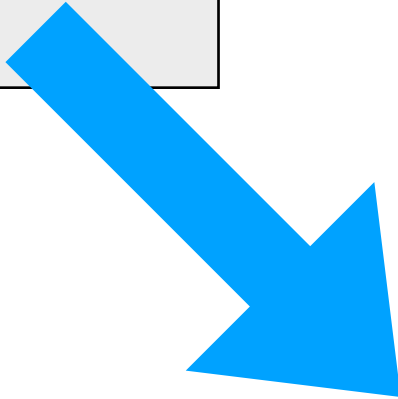
which refactor
is correct?

hint: `or2(False, True)`



```
def or2(cond1, cond2):  
    rv = False  
    rv = rv or cond1  
    rv = rv or cond2  
    return rv
```

A

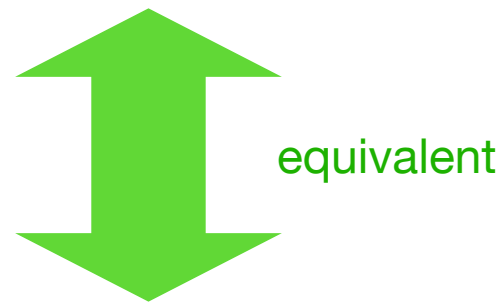


```
def or2(cond1, cond2):  
    if cond1:  
        return cond2  
    else:  
        return False
```

B

Refactor Exercise 1

```
return b1 or b2 or b3 or ... or bN
```



```
rv = False  
rv = rv or b1  
rv = rv or b2  
rv = rv or b3  
...  
rv = rv or bN
```


Lesson: with "or", it only takes one to flip the whole thing True!

Refactor Exercise 2

```
def and2(cond1, cond2):  
    return cond1 and cond2
```

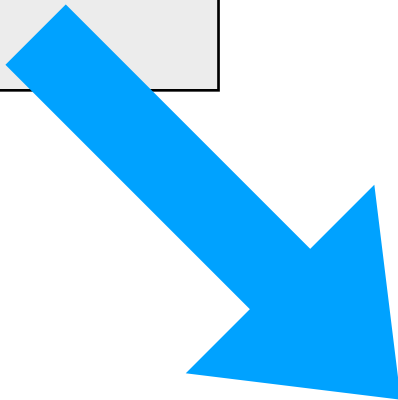
which refactor
is correct?

hint: `and2(True, True)`



```
def and2(cond1, cond2):  
    rv = False  
    rv = rv and cond1  
    rv = rv and cond2  
    return rv
```

A

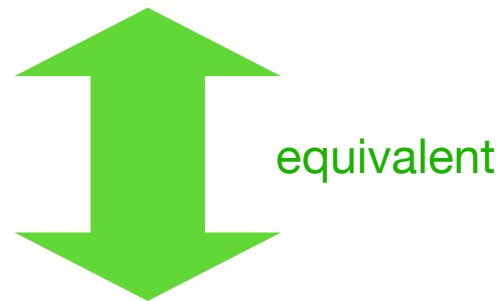


```
def and2(cond1, cond2):  
    if cond1:  
        return cond2  
    else:  
        return False
```

B

Refactor Exercise 2

```
return b1 and b2 and b3 and ... and bN
```



```
if b1:  
    return b2 and b3 and ... and bN  
else:  
    return False
```

Lesson: with "and", the first one can make the whole thing False!

Refactor Exercise 3

which refactor
is correct?

hint: `fix(False, False)`

```
def fix(moves, should):
    if moves:
        if should:
            return "good"
        else:
            return "duct tape"
    else:
        if should:
            return "WD-40"
        else:
            return "good"
```

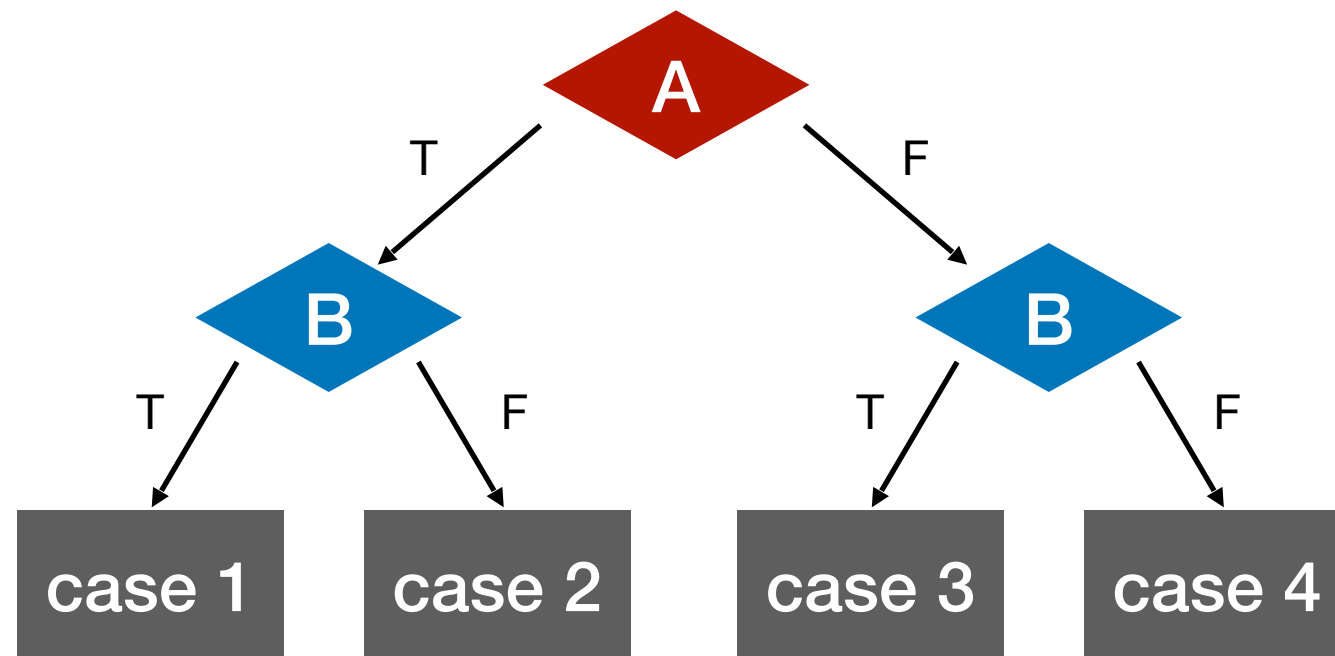
A

```
def fix(moves, should):
    if moves and not should:
        return "duct tape"
    elif not moves and should:
        return "WD-40"
    elif moves and should:
        return "good"
    elif not moves and not should:
        return "good"
```

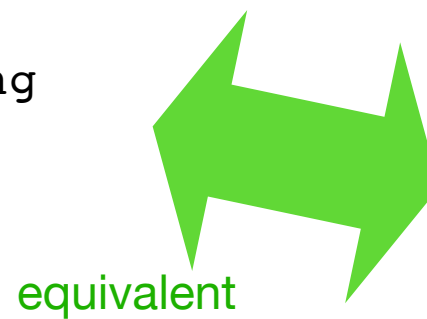
B

```
def fix(moves, should):
    if should:
        if moves:
            return "duct tape"
        else:
            return "good"
    else:
        if moves:
            return "good"
        else:
            return "duct tape"
```

Refactor Exercise 3

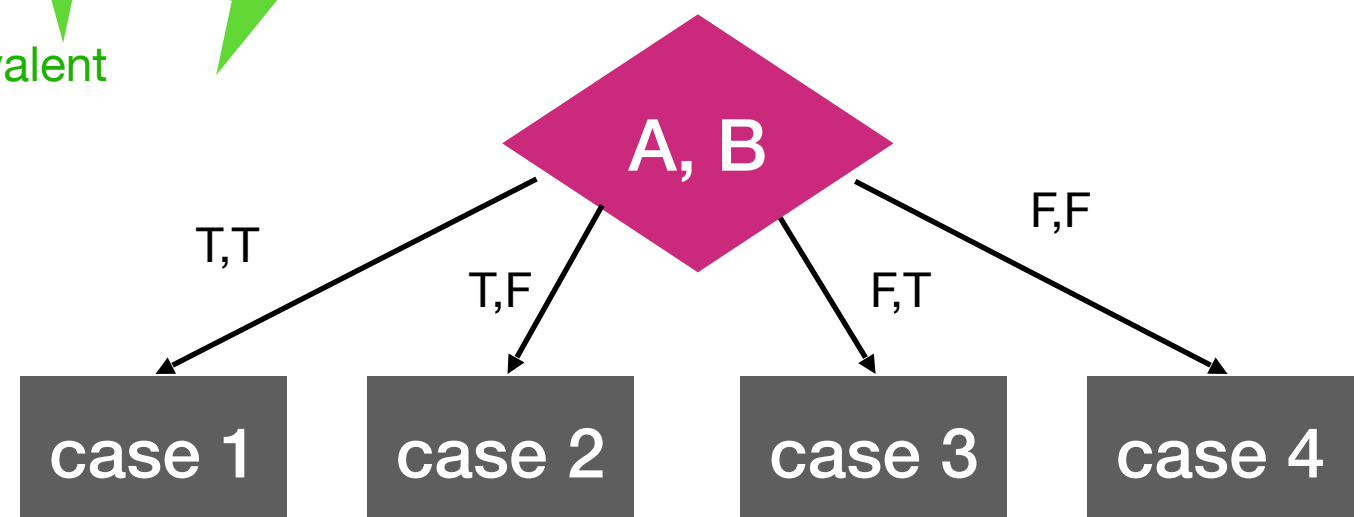


Option 1: Nesting



equivalent

Option 2: Chaining

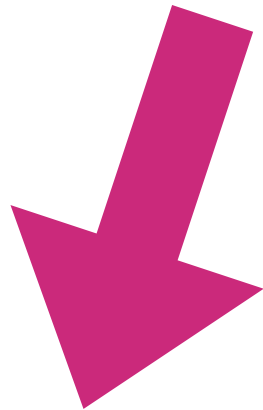


Lesson: when handling combinations of booleans, you can either do either (a) nesting or (b) chaining with and

Refactor Exercise 4

which refactor
is correct?

hint: `is_220(2, 2, 0)`



```
def is_220(a, b, c):  
    if a==2:  
        if c==0:  
            if b==2:  
                return True  
    return False
```

A

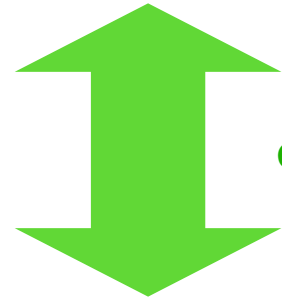


```
def is_220(a, b, c):  
    if a==2 or b==2 or c==0:  
        return False  
    return True
```

B

Refactor Exercise 4

```
return b1 and b2 and b3 and ... and bN
```



equivalent

```
if b1:
    if b2:
        if b3:
            ...
            if bN:
                return True
return False
```

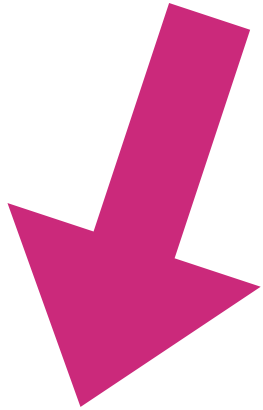
Lesson: nesting a lot of if's inside each other is equivalent to and'ing all the conditions

```
def is_220(a, b, c):  
    return a==2 and b==2 and c==0
```

Refactor Exercise 5

which refactor
is correct?

hint: `is_220(2, 2, 1)`



```
def is_220(a, b, c):  
    if a==2:  
        return True  
    if b==2:  
        return True  
    if c==0:  
        return True  
    return False
```

A

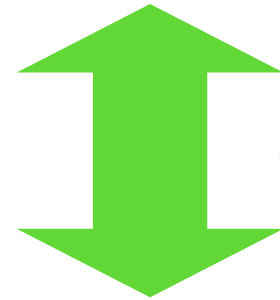


```
def is_220(a, b, c):  
    if a!=2:  
        return False  
    if b!=2:  
        return False  
    if c!=0:  
        return False  
    return True
```

B

Refactor Exercise 5

```
return b1 and b2 and b3 and ... and bN
```



equivalent

```
if not b1:
    return False
if not b2:
    return False
if not b3:
    return False
...
if not bN:
    return False
return True
```

Lesson: checking if everything is True can be translated
to seeing if we can find anything False