

# [220 / 319] Iteration 2

Meena Syamkumar  
Andy Kuemmel  
Cole Nelson

## Readings:

Chapter 2 of Sweigart book  
Chapter 6.4 of Python for Everybody

Due: Quiz3

Due: Student Information Survey

# Learning Objectives Today

Nested loops tracing

Chapter 7 of Think Python

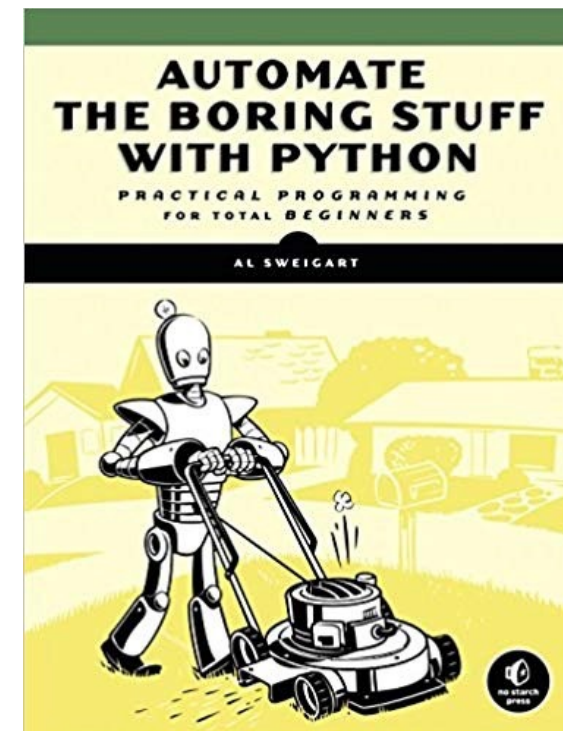
Understanding **break** and **continue**

- Syntax
- Control flow
- Use cases

Chapter 2 of Sweigart  
(great recap so far)

Nested loops tracing

- Interaction with break/continue



<http://automatetheboringstuff.com/chapter2/>

# Today's Outline

Design Patterns

Worksheet

Break

Continue

Nesting

# Design Patterns (outside Programming)

## Overview [\[edit\]](#)

---

The **five-paragraph essay** is a form of [essay](#) having five [paragraphs](#):

- 1st** • one introductory paragraph,
- 3rd** • three body paragraphs with support and development, and
- 2nd** • one concluding paragraph.

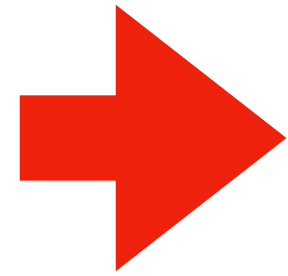
[wikipedia]

somebody familiar with this  
structure might skip around

there are many similarities between  
reading/writing code and essays

# Design Patterns

i	1 3
n	2 4



```
i = 1
while i <= 30:
    n = i * 2
    print(n)
    i += 1
```

## Output

2

4

...

When you ask a programmer what a piece of code does, what do they look at, and in what order?

**Way I: walk through in order (never a bad option)**

# Design Patterns

experienced coders will focus in  
on everything about “i” first  
because that is in the loop condition

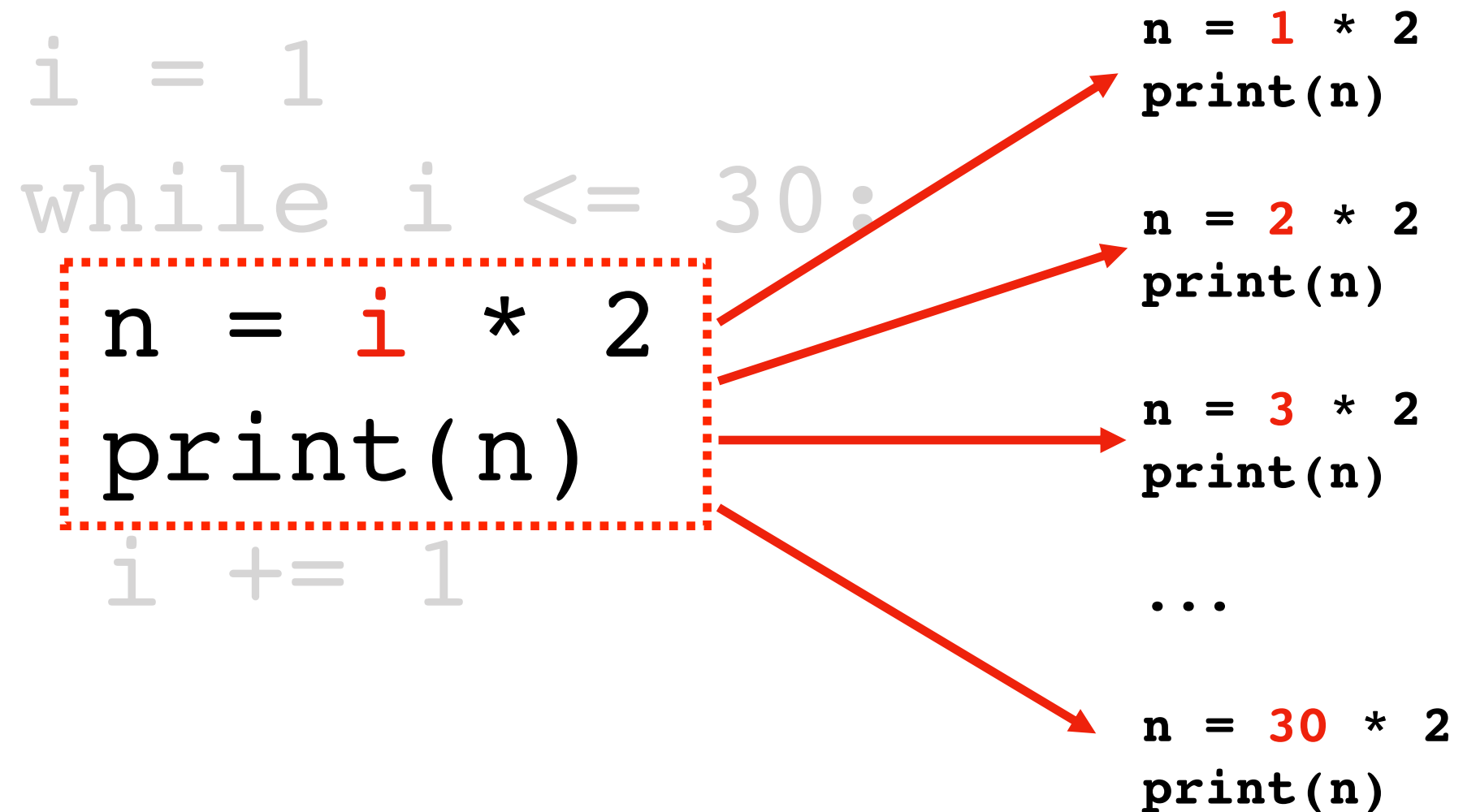
```
i = 1
while i <= 30:
    n = i * 2
    print(n)
    i += 1
```

When you ask a programmer what a piece of code  
does, what do they look at, and in what order?

**Way 2: knowing that certain code is written again  
and again, look for common patterns to break it down**

**Observation:** loop will run with values of i of: 1 to 30

# Design Patterns



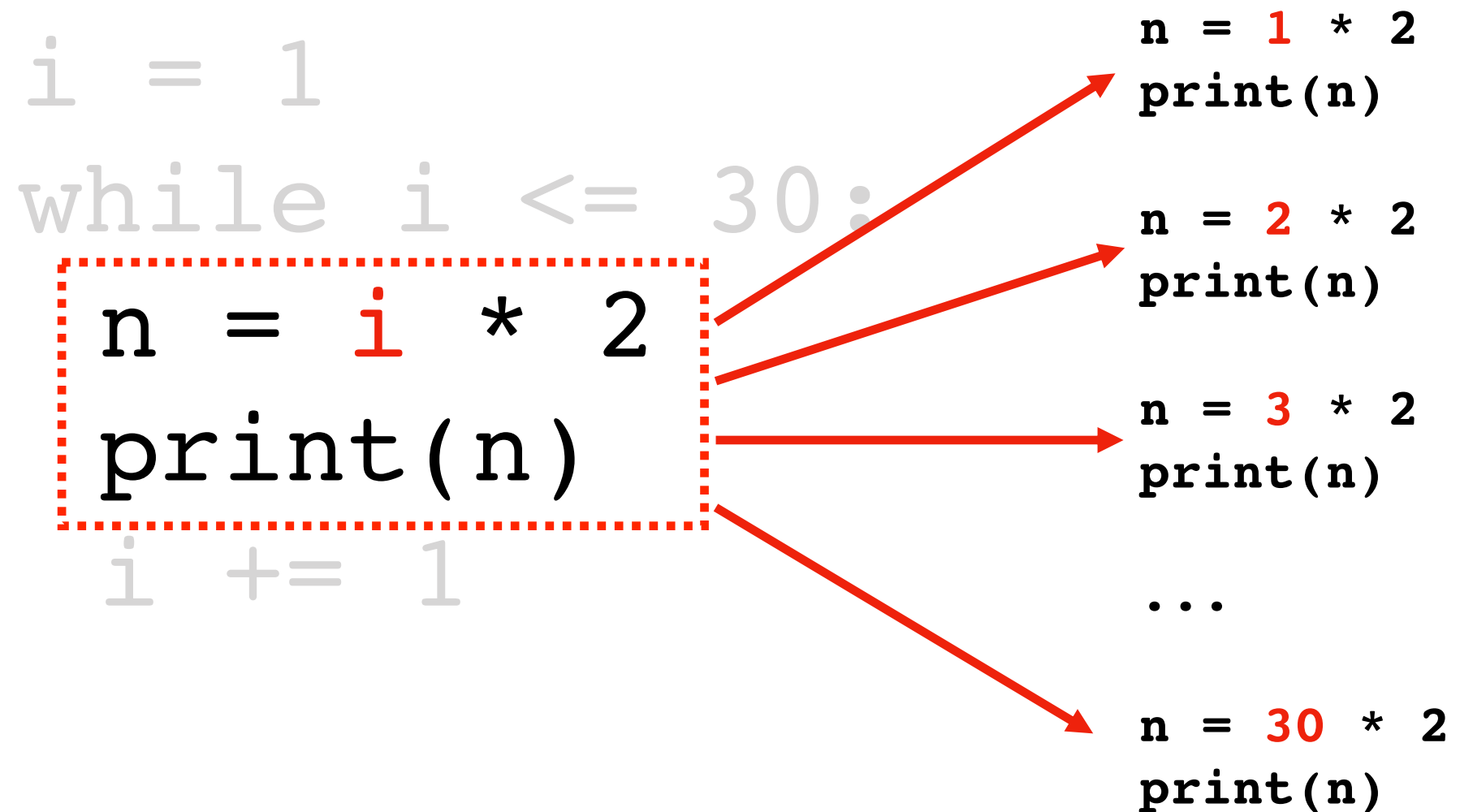
When you ask a programmer what a piece of code does, what do they look at, and in what order?

**Observation:** highlighted code runs 30 times, with `i` values of 1 through 30

# Design Patterns

## Output

2  
4  
6  
8  
...  
56  
58  
60



When you ask a programmer what a piece of code does, what do they look at, and in what order?

**Conclusion:** the code prints 2, 4, 6, ..., 58, 60



# Design Pattern I: do something N times

```
i = 1  
while i <= N:
```

**Option A**

fill in with specifics here

```
i += 1
```

1, 2, 3, ..., N

```
i = 0  
while i < N:
```

**Option B**

fill in with specifics here

```
i += 1
```

0, 1, 2, ..., N-1

# Design Pattern 2: do something with all data

```
i = 0  
while i < N:
```

fill in with specifics here

```
i += 1
```

## Functions:

count\_rows()  
get\_population(index)  
...

index 0

index 1

State	Population	Area
WI	5.795	...
CA	39.54	...
MN	5.577	...
...	...	...

# Design Pattern 2: do something with all data

```
i = 0
```

```
while i < count_rows():
```

```
    pop = get_population(i)
```

fill in with specifics here

```
    i += 1
```

assumes we  
use 0 for first row

## Functions:

`count_rows()`

`get_population(index)`

...

State	Population	Area
WI	5.795	...
CA	39.54	...
MN	5.577	...
...	...	...

# Design Pattern 3: do something until the end

```
while has_more():  
    data = get_next()
```

fill in with specifics here

People creating functions/modules for other programmers to use will often have functions for checking if there is more data and for getting the data one piece at a time

# Today's Outline

Design Patterns

Worksheet

- Problem 1
- Problem 2

Break

Continue

Nesting

## Problem 1: counting

```
countdown = 5
while countdown > 1:
    print(countdown)
    countdown -= 1
```

**countdown**

**5**

**4**

**3**

**2**

## Problem 2: loops inside loops

```
i = 1
while i <= 3:
    j = 1
    while j <= i:
        print(i)
        j += 1
    print( 'END' )
    i += 1
```

## Problem 2: loops inside loops

```
i = 1
while i <= 3:
    j = 1
    while j <= i:
        print(i)
        j += 1
    print( 'END' )
    i += 1
```

i	j
1	1
2	1
2	2
3	1
3	2
3	3

**Output**

```
1
END
2
2
END
3
3
3
END
```



# Today's Outline

Design Patterns

Worksheet

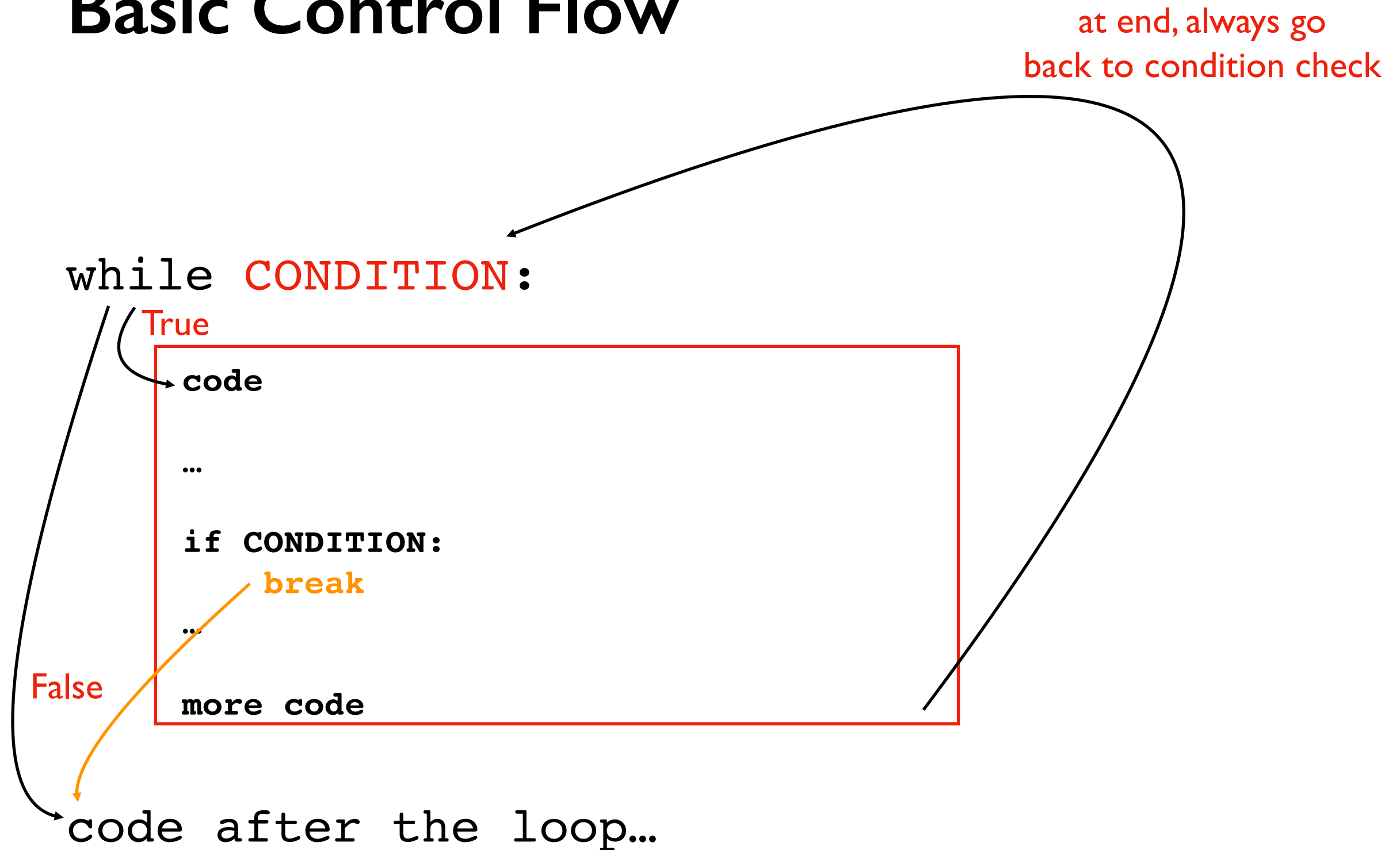
**Break**

Don't get too excited,  
only the loops get a break!

Continue

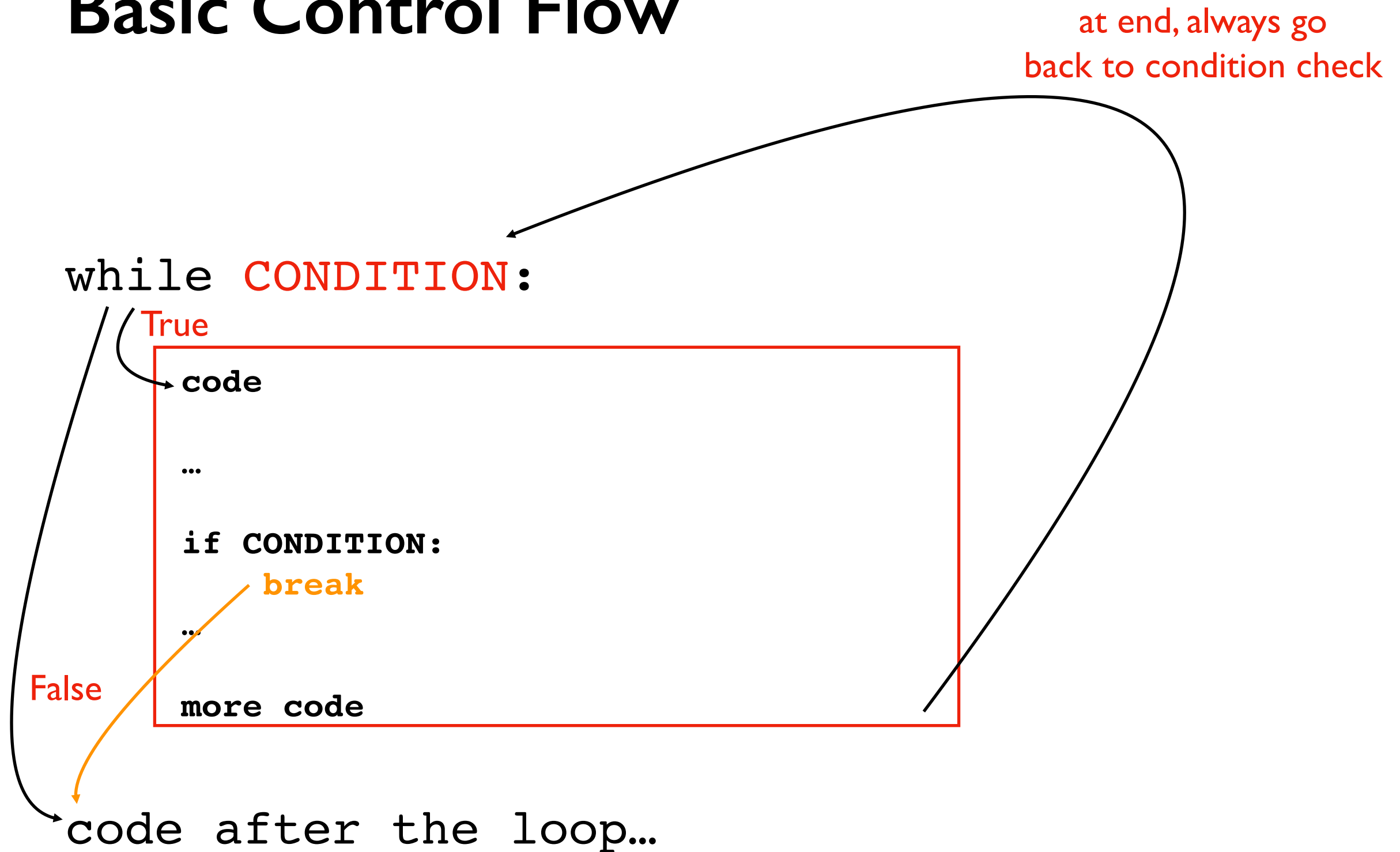
Nesting

# Basic Control Flow



Just like `return` immediately exits a function,  
`break` immediately exits a loop

# Basic Control Flow



Usage: Commonly used when we're searching through many things.  
Allows us to stop as soon as we find what we want.

# Demo: Prime Search Program

Goal: answer whether a range of numbers contains a prime

## Input:

- Start of range
- End of range



## Output:

- Yes or no

## Examples:

14 to 16 => NO (because 14, 15, and 16 are all not prime)

10 to 12 => YES (because 11 is prime)

## Problem 3: can we have a *break*, please?

```
num = 0
while num < 500:
    num += 100
    print(str(num) + "?")
    if num == 300:
        break
    print('YES')
```

num	inside sandwich
0	100
100	200
200	300
300	400
400	500

output  
100?  
YES  
200?  
YES  
300?

# Today's Outline

Design Patterns

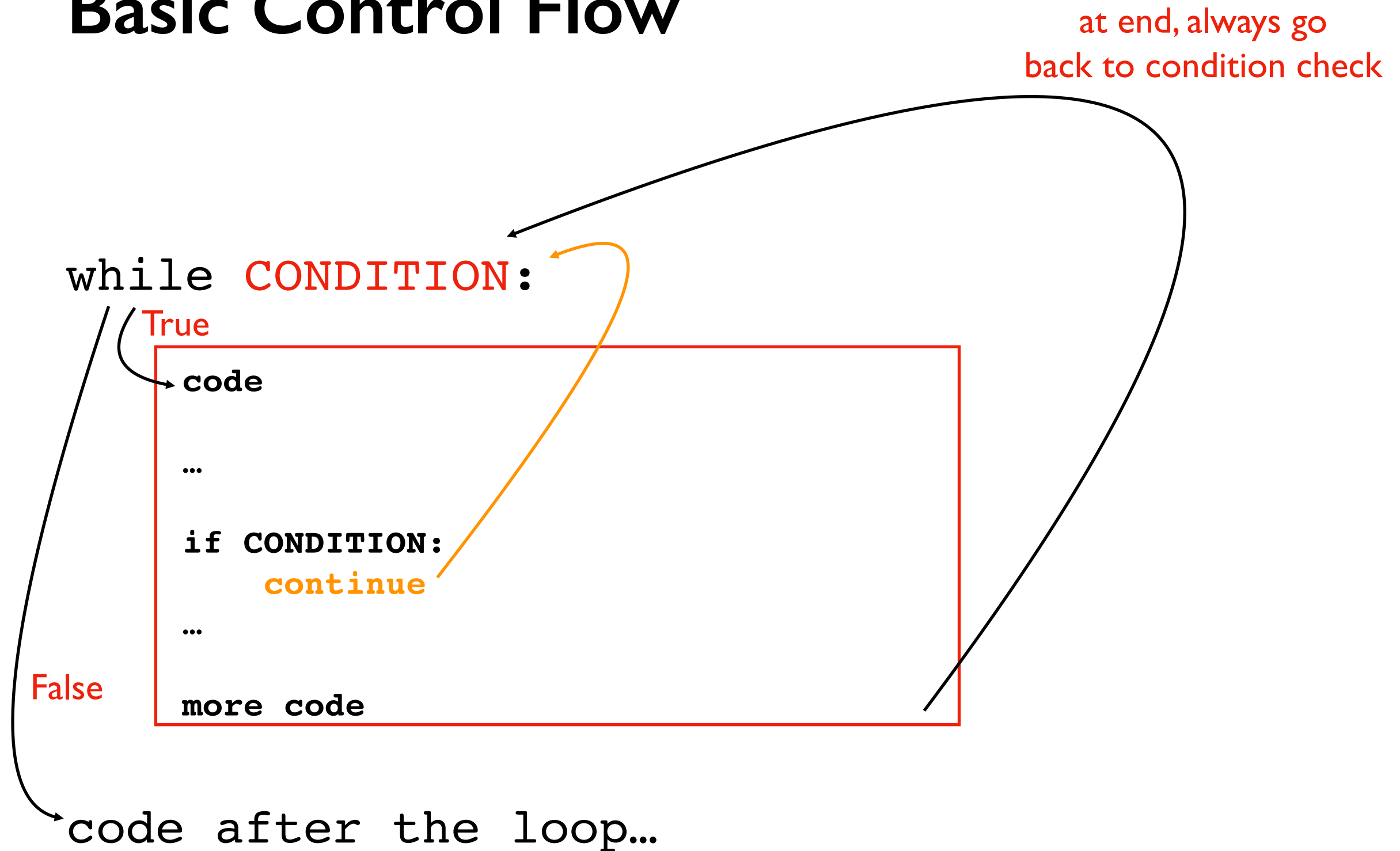
Worksheet

Break

Continue

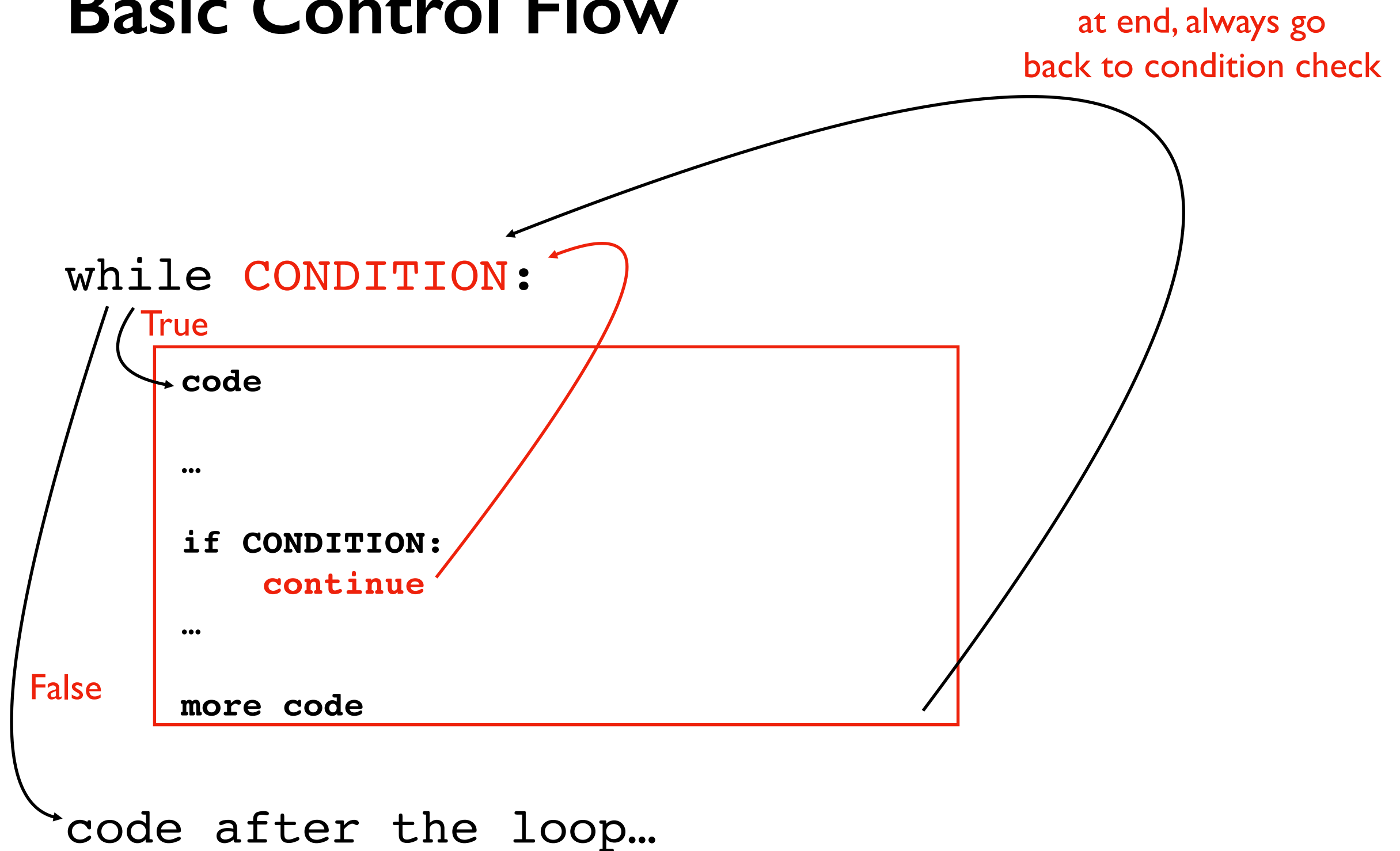
Nesting

# Basic Control Flow



`continue` immediately stops current iteration and goes back to the condition, without executing the "more code part, potentially to start another iteration

# Basic Control Flow



Usage: commonly used to skip over values we want to ignore



# Demo: Average Score

Goal: keep a running average of user-provided scores

## Input:

- “q” for quit (keep running until this)
- a score in the 0 to 100 range

## Output:

- Recompute average and print after each new number

## Example:

enter a score (or q for exit): **50**

avg is 50

enter a score (or q for exit): **110**

bad input, skipping!

enter a score (or q for exit): **q**

exiting

Twist: use “continue” to skip over inputs not in the 0 to 100 range

# Today's Outline

Design Patterns

Worksheet

Break

Continue

Nesting

## Problem 4: we must *continue* practicing loops!

```
num = 0
while num < 500:
    num += 100
    print(str(num) + "?")
    if num == 300:
        continue
    print('YES')
```

num	inside sandwich
0	100
100	200
200	300
300	400
400	500

output

100?  
YES  
200?  
YES  
300?  
400?  
YES  
500?  
YES

# Nested loops

```
while CONDITION_A:
```

```
# more code
```

```
while CONDITION_B:
```

```
# more code
```

```
if CONDITION_C:
```

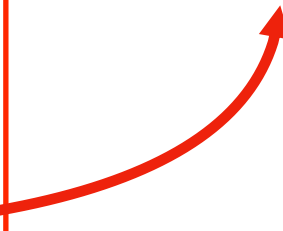
```
continue
```

```
# more code
```

```
# more code
```

```
# code outside any loop
```

where does this  
jump back to?



# Nested loops

```
while CONDITION_A:
```

```
# more code
```

```
while CONDITION_B:
```

```
# more code
```

```
if CONDITION_C:
```

```
continue
```

```
# more code
```

```
# more code
```

```
# code outside any loop
```

continue and break  
always apply to the  
inner loop in Python

# Nested loops

```
while CONDITION_A:
```

```
# more code
```

```
while CONDITION_B:
```

```
# more code
```

```
if CONDITION_C:
```

```
break
```

```
# more code
```

```
# more code
```



```
# code outside any loop
```

# Worksheet Problems