

# [220] Iteration 2

Meena Syamkumar  
Mike Doescher

- **Exam I Friday evening – single page notes allowed**
- **Partner matching**

**Cheaters caught: 0**  
**Piazza Enrollment 440 / 446**

# Learning Objectives Today

## Understand **break**

- Syntax
- Control flow
- Use cases

## Understand **continue**

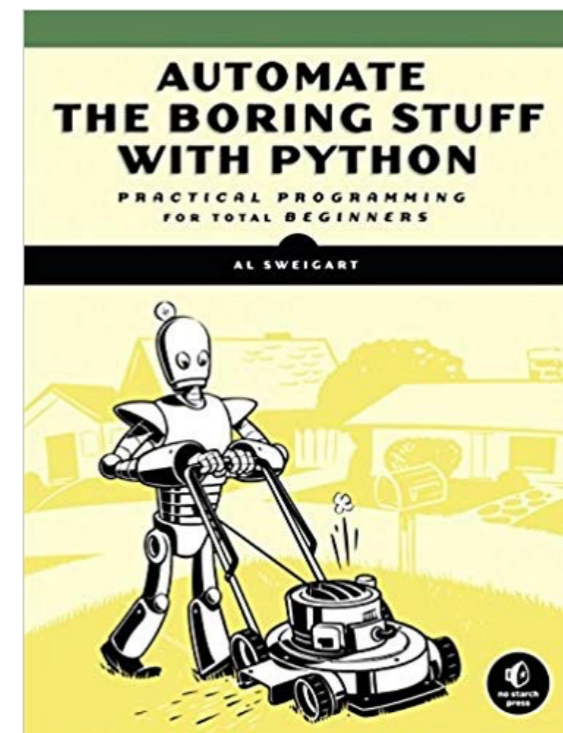
- Syntax
- Control flow
- Use cases

## Nested loops

- Interaction with break/continue

Chapter 7 of Think Python

Chapter 2 of Sweigart  
(great recap so far)



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

# Today's Outline

Design Patterns

Worksheet

Break

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

Continue

Nesting

# Design Patterns (outside Programming)

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

---

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

- one introductory paragraph,
- three body paragraphs with support and development, and
- one concluding paragraph.

[wikipedia]

# Design Patterns (outside Programming)

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

---

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

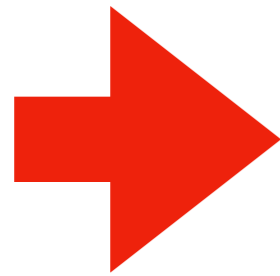
- 1st** • one introductory paragraph,
- 3rd** • three body paragraphs with support and development, and
- 2nd** • one concluding paragraph.

[wikipedia]

somebody familiar with this structure might skip around

there are many similarities between reading/writing code and essays

# Design Patterns



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

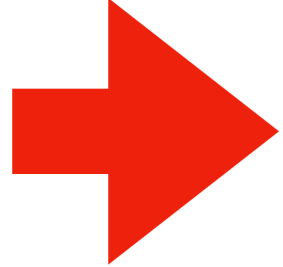
*When you ask a programmer what a piece of code does, what do they look at, and in what order?*

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

# Design Patterns

i

1



```
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?

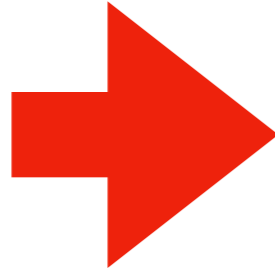
**Way I: walk through in order (never a bad option)**

# Design Patterns

i

1

```
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?

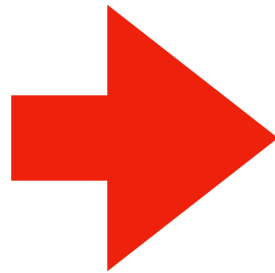
**Way I: walk through in order (never a bad option)**



# Design Patterns

i	1
n	2

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

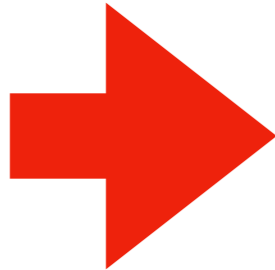
# Design Patterns

i	1
n	2

Output

2

```
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?

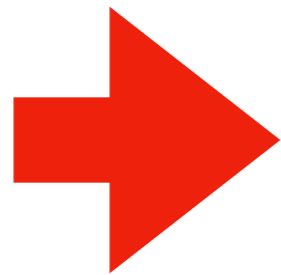
**Way I: walk through in order (never a bad option)**

# Design Patterns

i	1 2
n	2

Output

2



```
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?

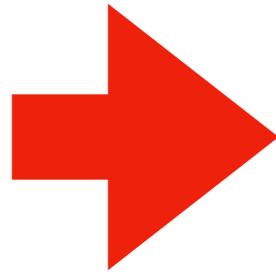
**Way I: walk through in order (never a bad option)**

# Design Patterns

i	1 2
n	2

Output

2



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

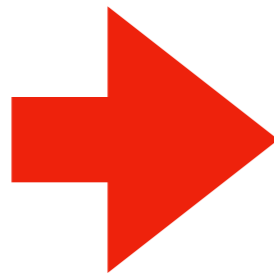
# Design Patterns

i	1 2
n	2 4

Output

2

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

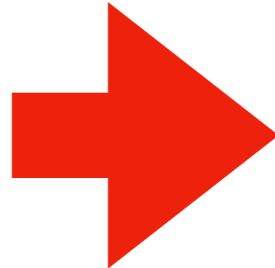
# Design Patterns

i	1 2
n	2 4

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

**Output**

2  
4

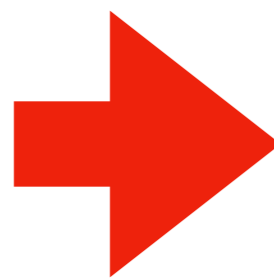


When you ask a programmer what a piece of code does, what do they look at, and in what order?

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

# Design Patterns

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

```
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**



# 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?

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

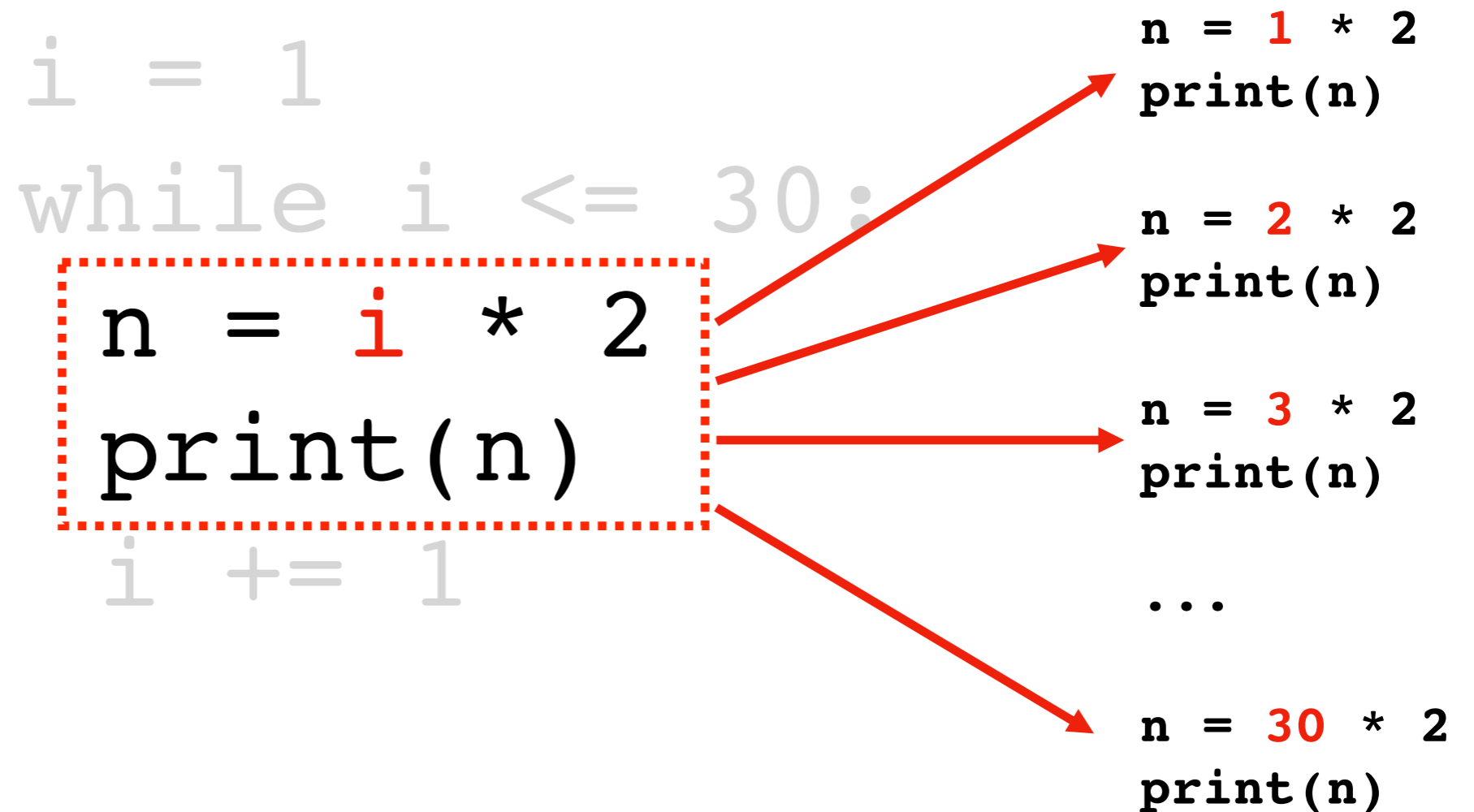
# Design Patterns

```
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?

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

# Design Patterns



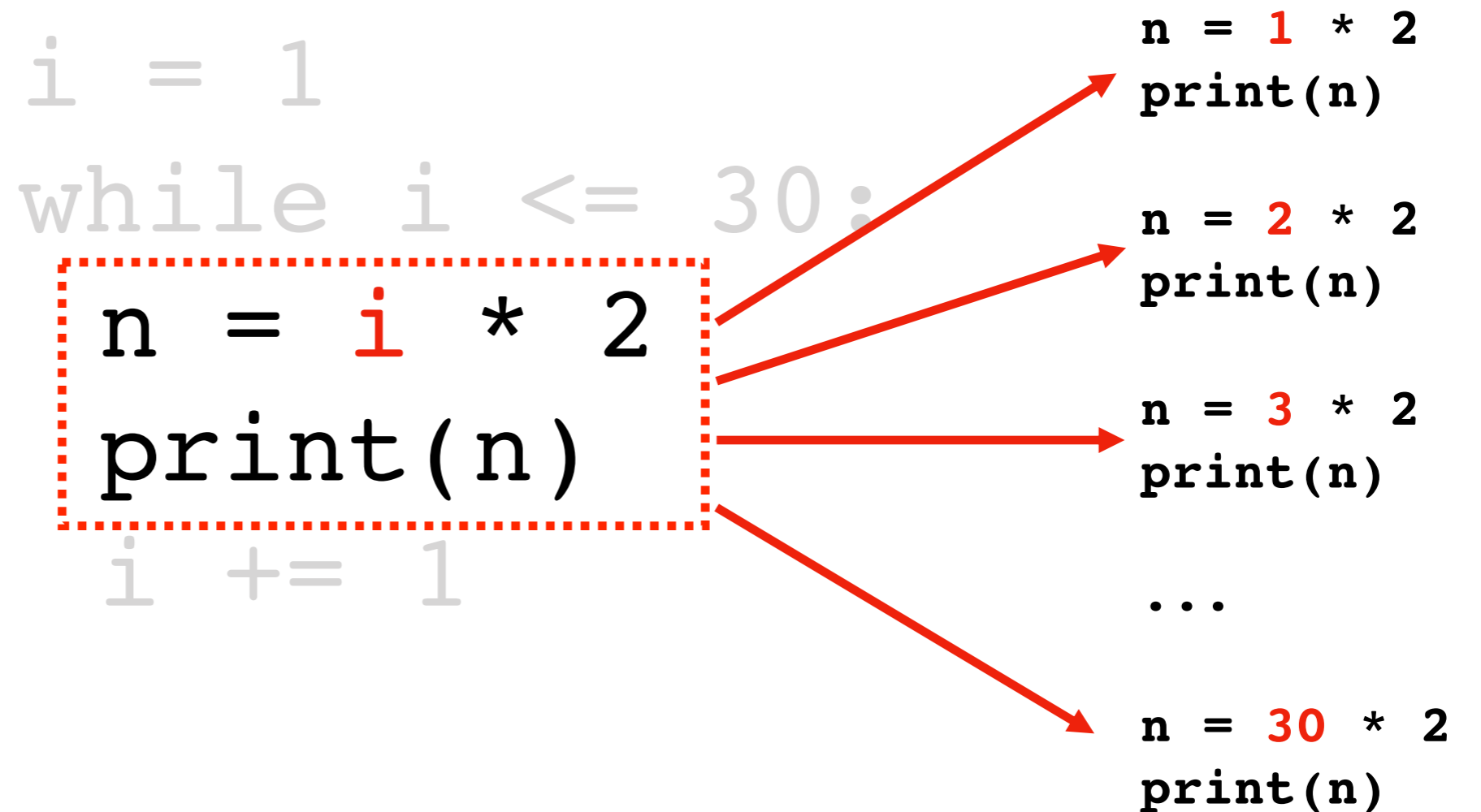
When you ask a programmer what a piece of code does, what do they look at, and in what order?

**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
```

**Option B**

# Design Pattern I: do something N times

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

**Option A**

fill in with specifics here

```
i += 1
```

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

**Option B**

fill in with specifics here

```
i += 1
```

# 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
```

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



# 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

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

# 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 I

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
```

## Functions:

`count_rows()`

`get_population(index)`

...

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)
```

assumes we  
use 0 for first row

fill in with specifics here

```
i += 1
```

## 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

# Today's Outline

Design Patterns

Worksheet

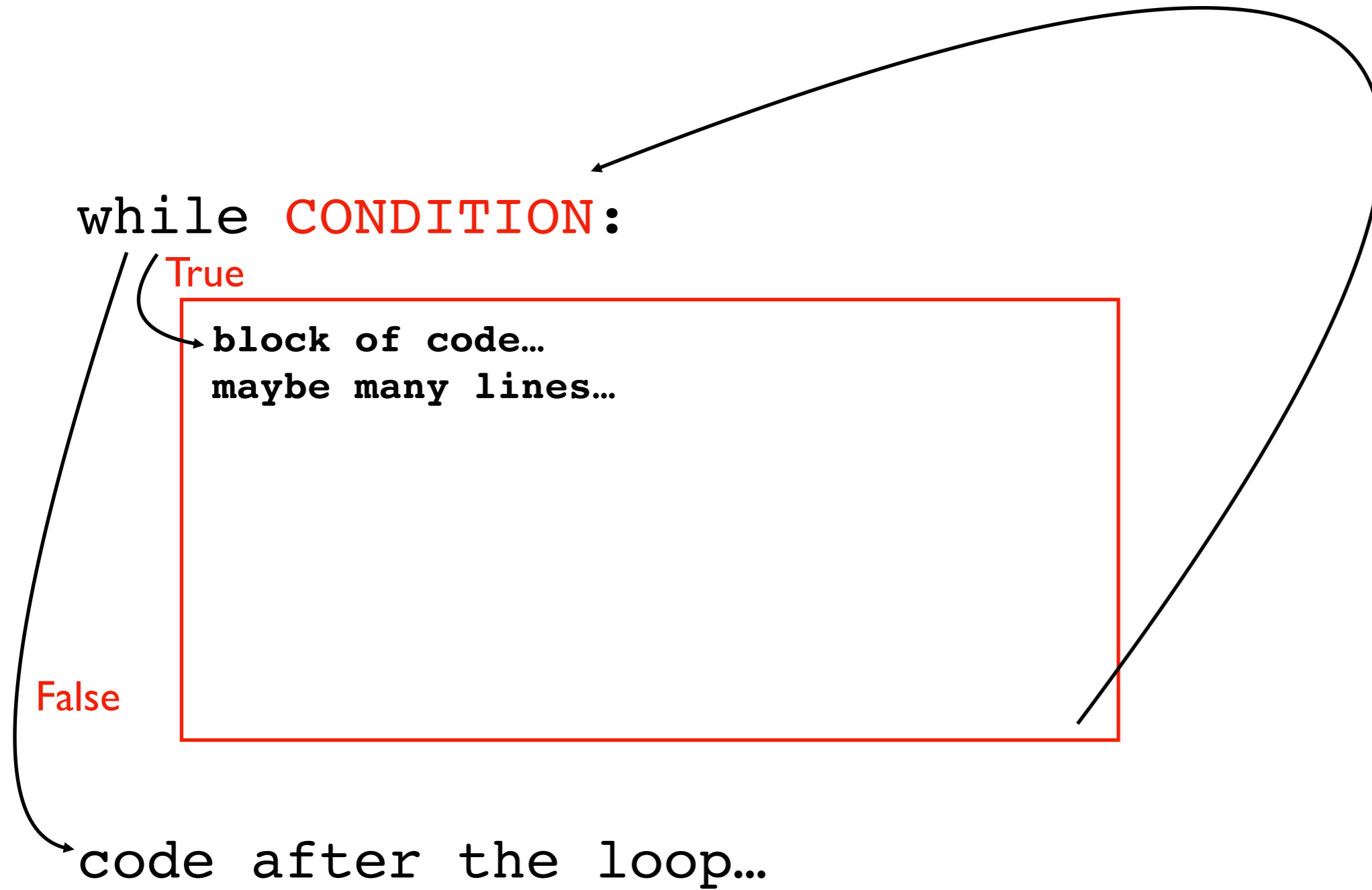
Break

Continue

Nesting

# Basic Control Flow

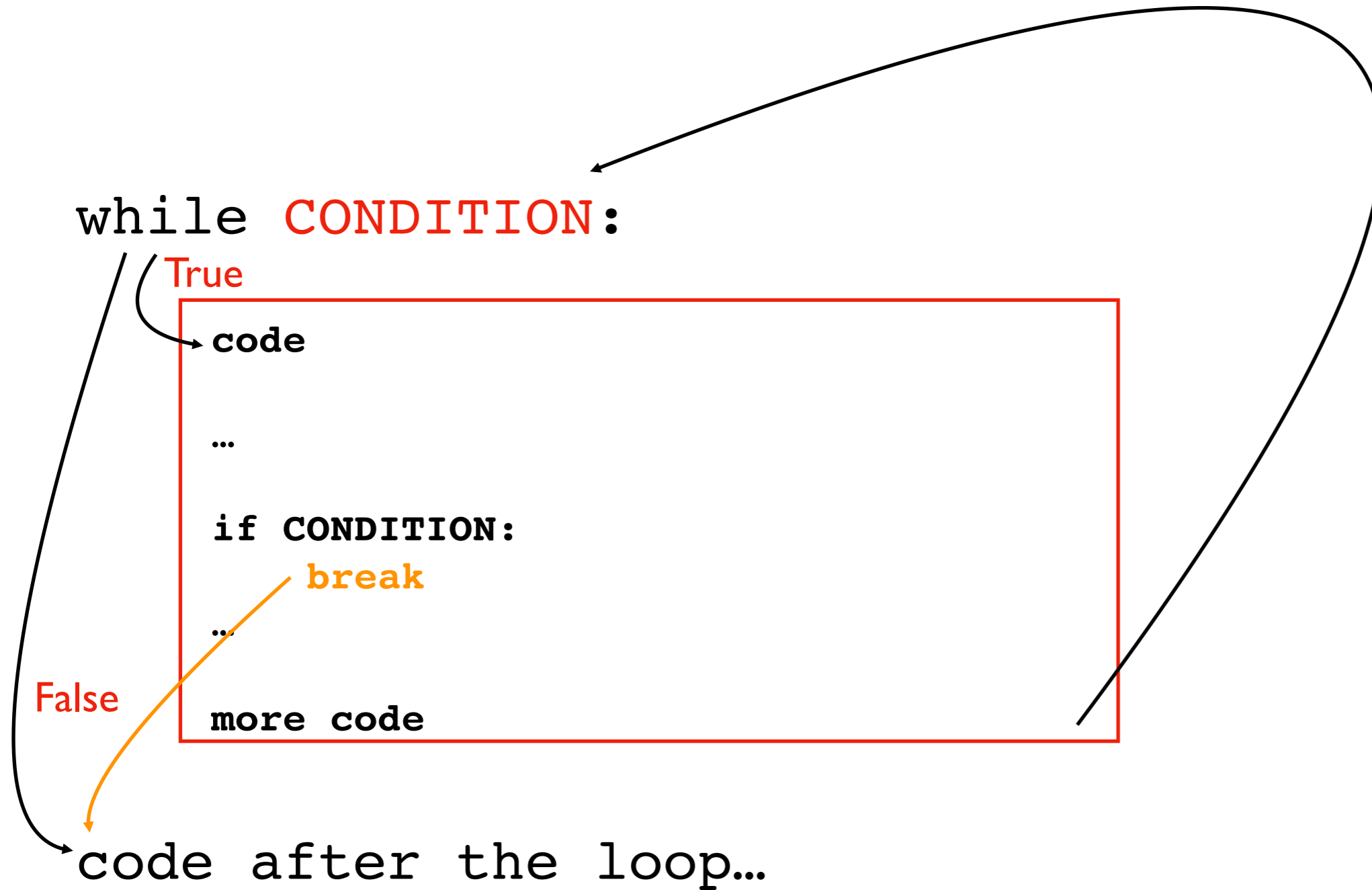
at end, always go  
back to condition check





# Basic Control Flow

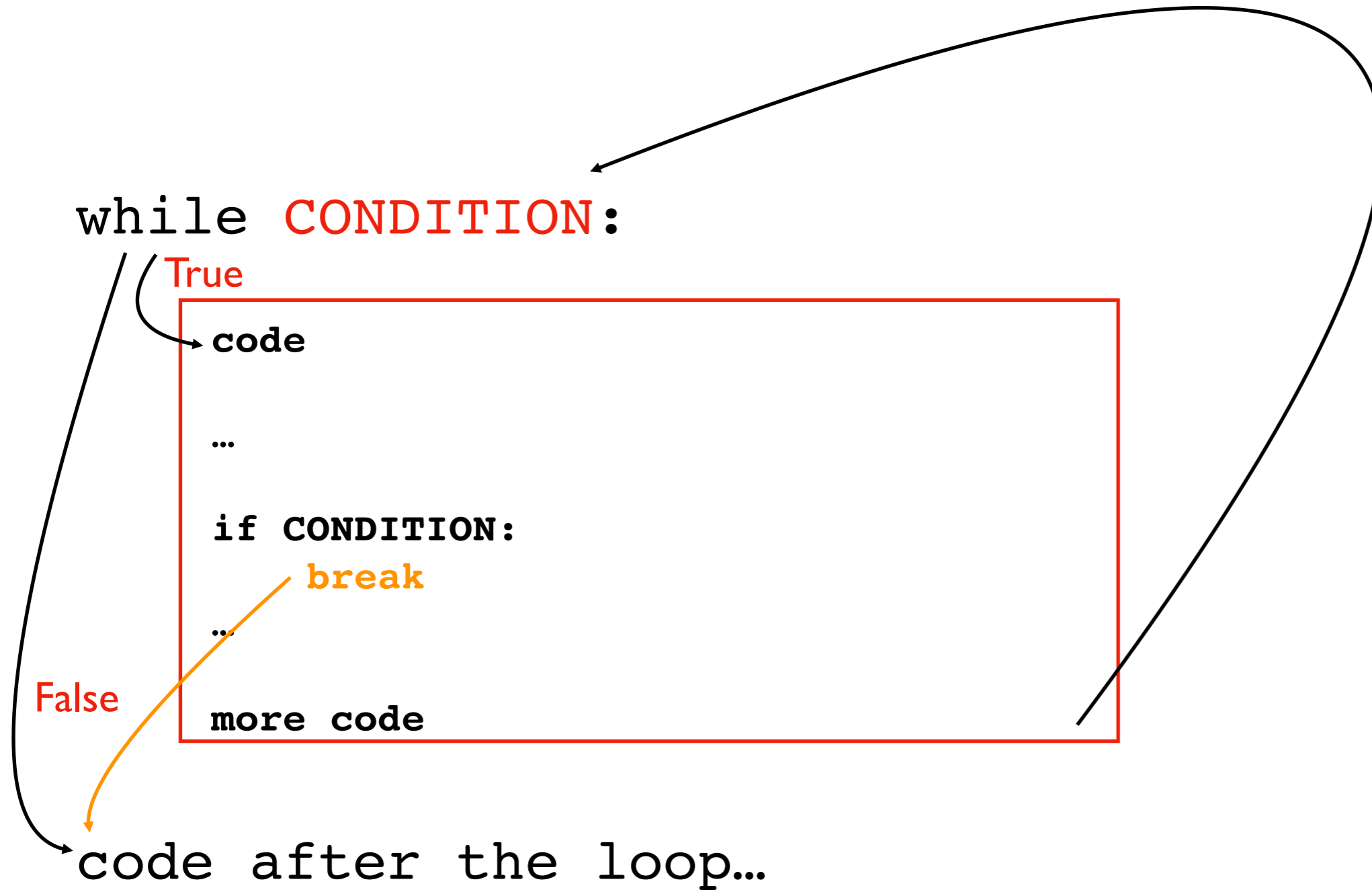
at end, always go  
back to condition check



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

# Basic Control Flow

at end, always go  
back to condition check



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)

# Today's Outline

Design Patterns

Worksheet

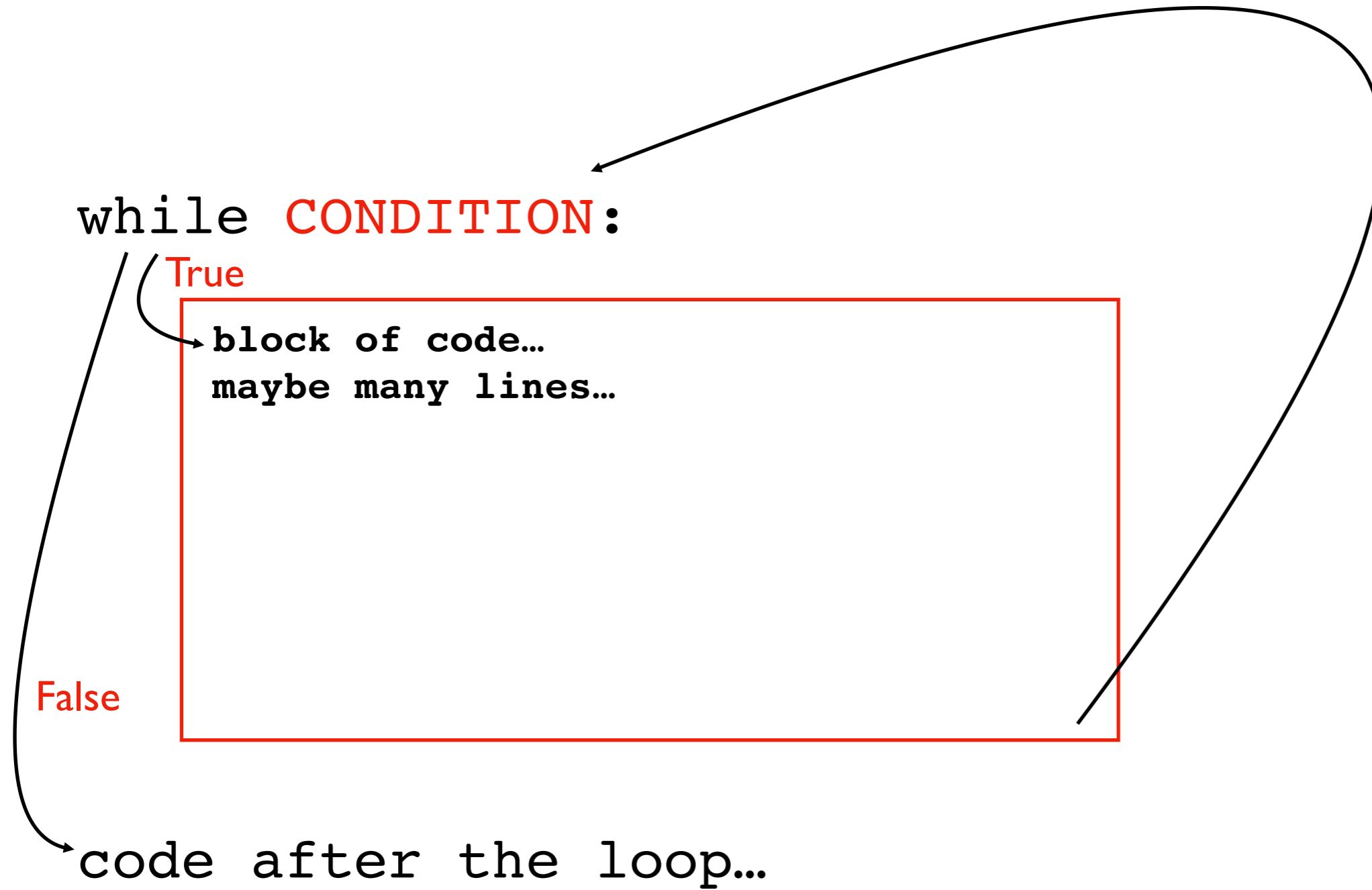
Break

**Continue**

Nesting

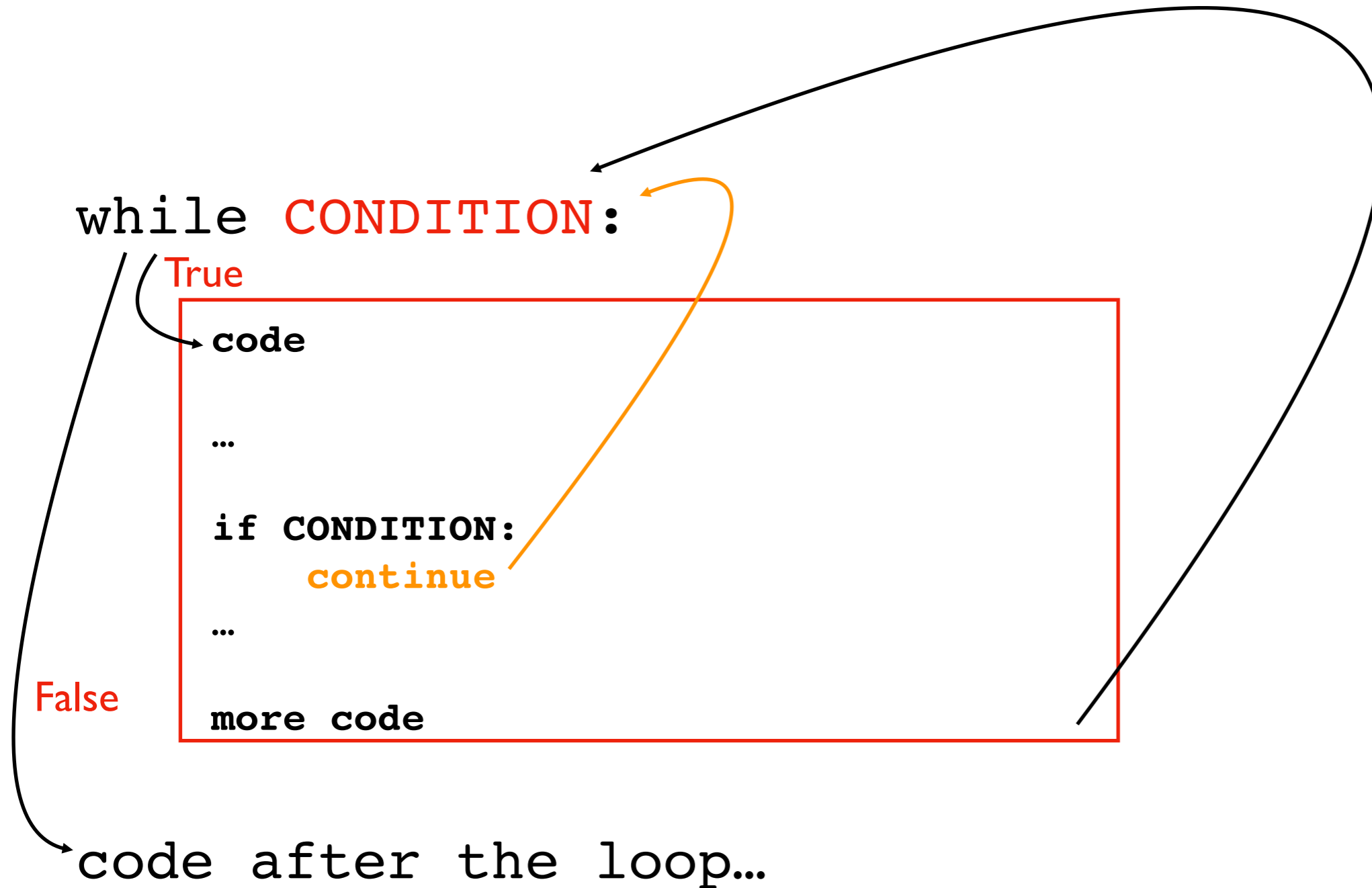
# Basic Control Flow

at end, always go  
back to condition check



# Basic Control Flow

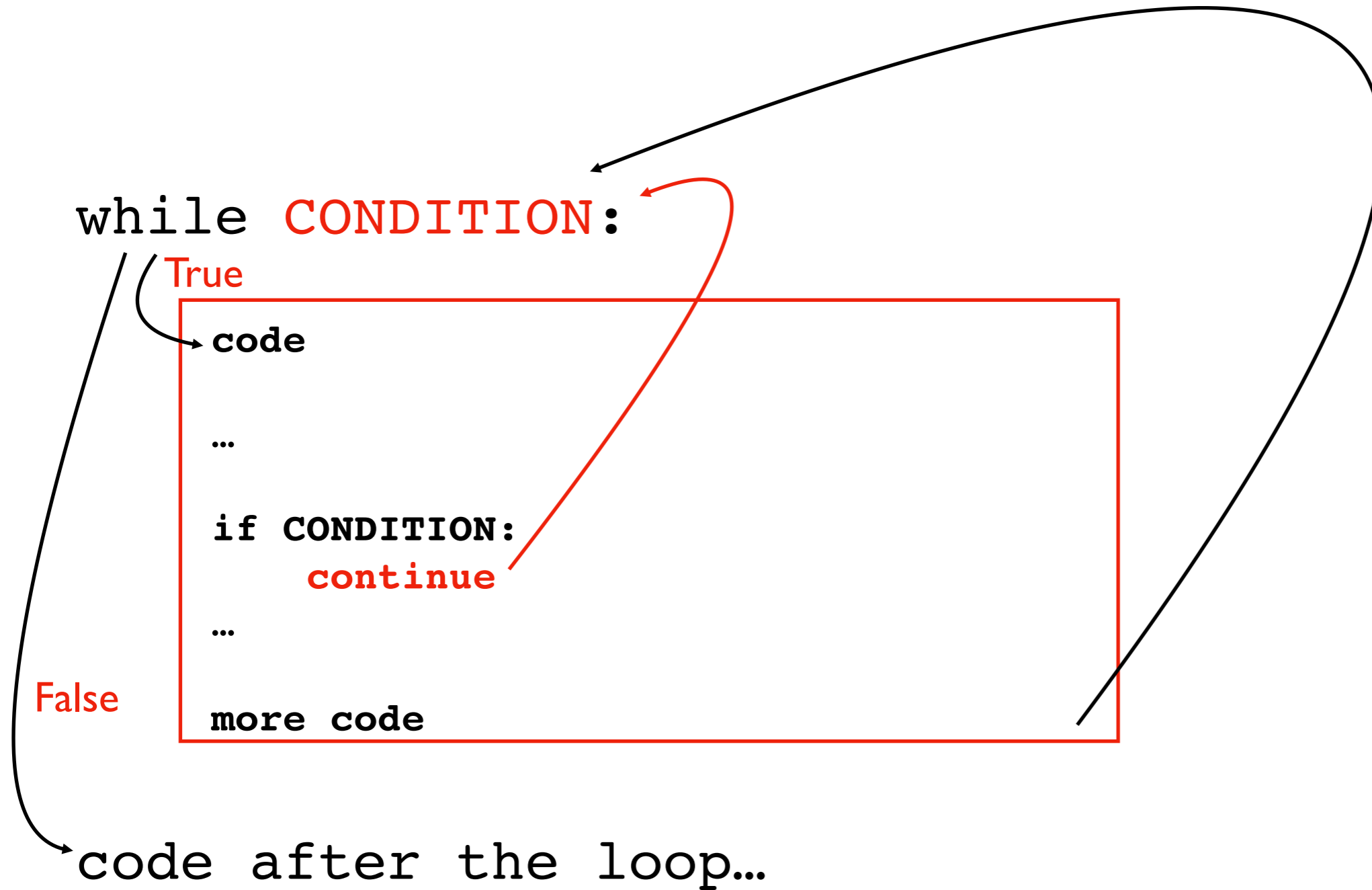
at end, always go  
back to condition check



`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

at end, always go  
back to condition check



**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**

# Nested loops

```
while CONDITION_A:
```

```
    # more code
```

```
        while CONDITION_B:
```

```
            # more code
```

```
                if CONDITION_C:
```

```
                    continue
```

```
                # more code
```

```
            # more code
```

```
    # code outside any loop
```

*how many blocks are there?*

# Nested loops

```
while CONDITION_A:
```

```
# more code
```

```
while CONDITION_B:
```

```
# more code
```

```
if CONDITION_C:
```

```
continue
```

```
# more code
```

```
# more code
```

```
# code outside any loop
```

# 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