[301] Advanced Iteration

Tyler Caraza-Harter

Learning Objectives Today

Understand “break”

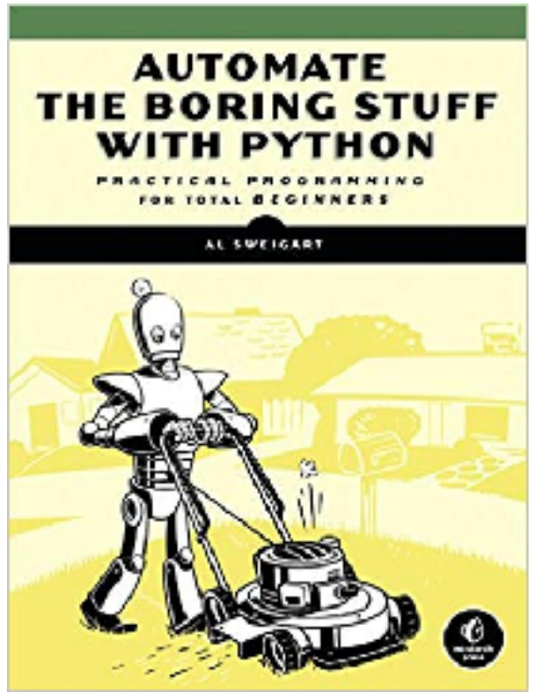
- Syntax
- Control flow
- Use cases

Chapter 7 of Think Python

Understand “continue”

- Syntax
- Control flow
- Use cases

Chapter 2 of Sweigart



Nested loops

- Interaction with break/continue

http://automatetheboringstuff.com/chapter2/

Today's Outline

Design Patterns

Worksheet

Break

Continue

Nesting

Today's Outline

Design Patterns

Worksheet

Break

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

Continue

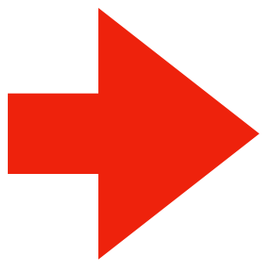
Nesting

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?

Design Patterns

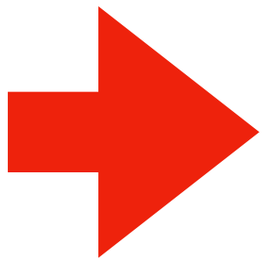


```
i = 1
while i <= 30:
    n = i * 2
    print(n)
    i += 1
```

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

Way 1: walk through in order (never a bad option)

Design Patterns

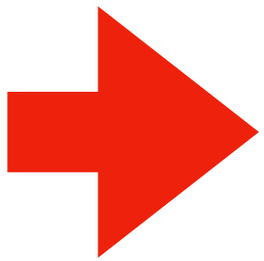


```
i = 1
while i <= 30:
    n = i * 2
    print(n)
    i += 1
```

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

Way 1: 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 1: 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 1: walk through in order (never a bad option)

Design Patterns

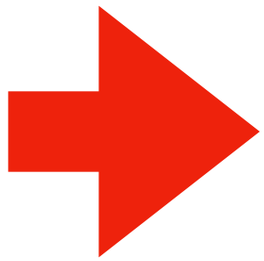
```
i = 1
while i <= 30:
  n = i * 2
  print(n)
  i += 1
```

Output
2

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

Way 1: walk through in order (never a bad option)

Design Patterns



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

Output
2

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

Way 1: walk through in order (never a bad option)

Design Patterns

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

Output
2

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

Way 1: walk through in order (never a bad option)

Design Patterns

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

Output
2

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

Way 1: walk through in order (never a bad option)

Design Patterns

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

Design Patterns

```
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 1: 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

```
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

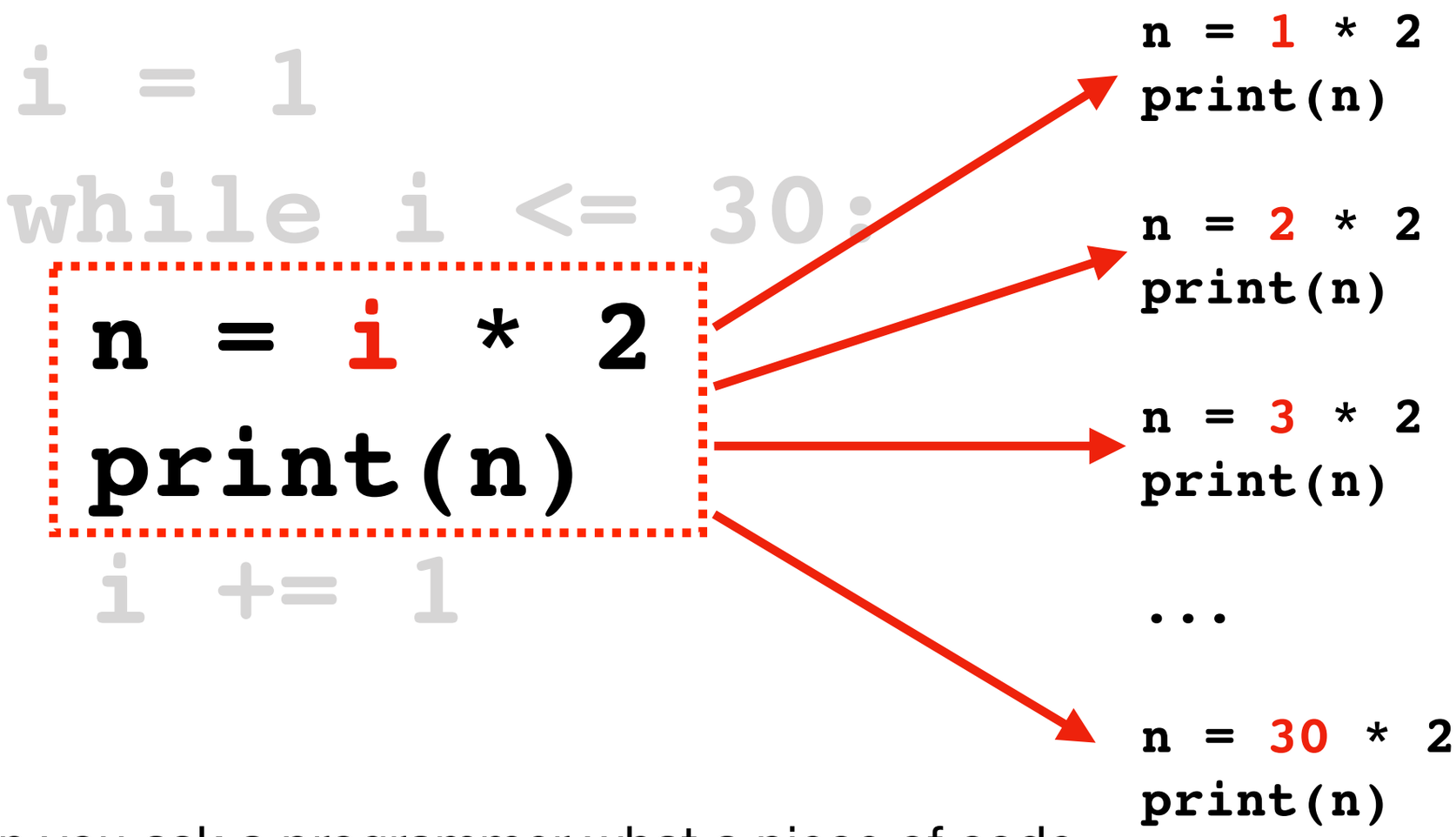
Design Patterns

```
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

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

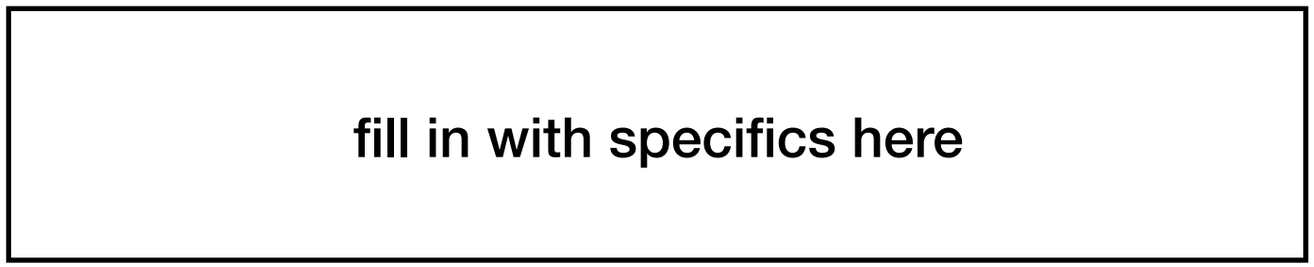
```
n = 1 * 2
print(n)

n = 2 * 2
print(n)

n = 3 * 2
print(n)

...

n = 30 * 2
print(n)
```

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 1: do something N times

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




```
    i += 1
```

Design Pattern 1: do something N times

```
i = 1
while i <= N:
    fill in with specifics here
    i += 1
```

Design Pattern 1: do something N times

Option A

```
i = 1
while i <= N:
    fill in with specifics here
    i += 1
```

Option B

```
i = 0
while i < N:
    fill in with specifics here
    i += 1
```

Design Pattern 1: do something N times

Option A

```
i = 1
while i <= N:
    fill in with specifics here
    i += 1
```

1, 2, 3, …, N

Option B

```
i = 0
while i < N:
    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	…	…
CA	…	…
MN	…	…
…	…	…

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)
…

State	Population	Area
WI	…	…
CA	…	…
MN	…	…
…	…	…

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	…	…
CA	…	…
MN	…	…
…	…	…

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)
…

State	Population	Area
WI	…	…
CA	…	…
MN	…	…
…	…	…

index 1

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	…	…
CA	…	…
MN	…	…
…	…	…

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	…	…
CA	…	…
MN	…	…
…	…	…

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

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

Like example when we add numbers from nums.txt

Today's Outline

Design Patterns

Worksheet
- Problem 1
- Problem 2

Break

Continue

Nesting

Today's Outline

Design Patterns

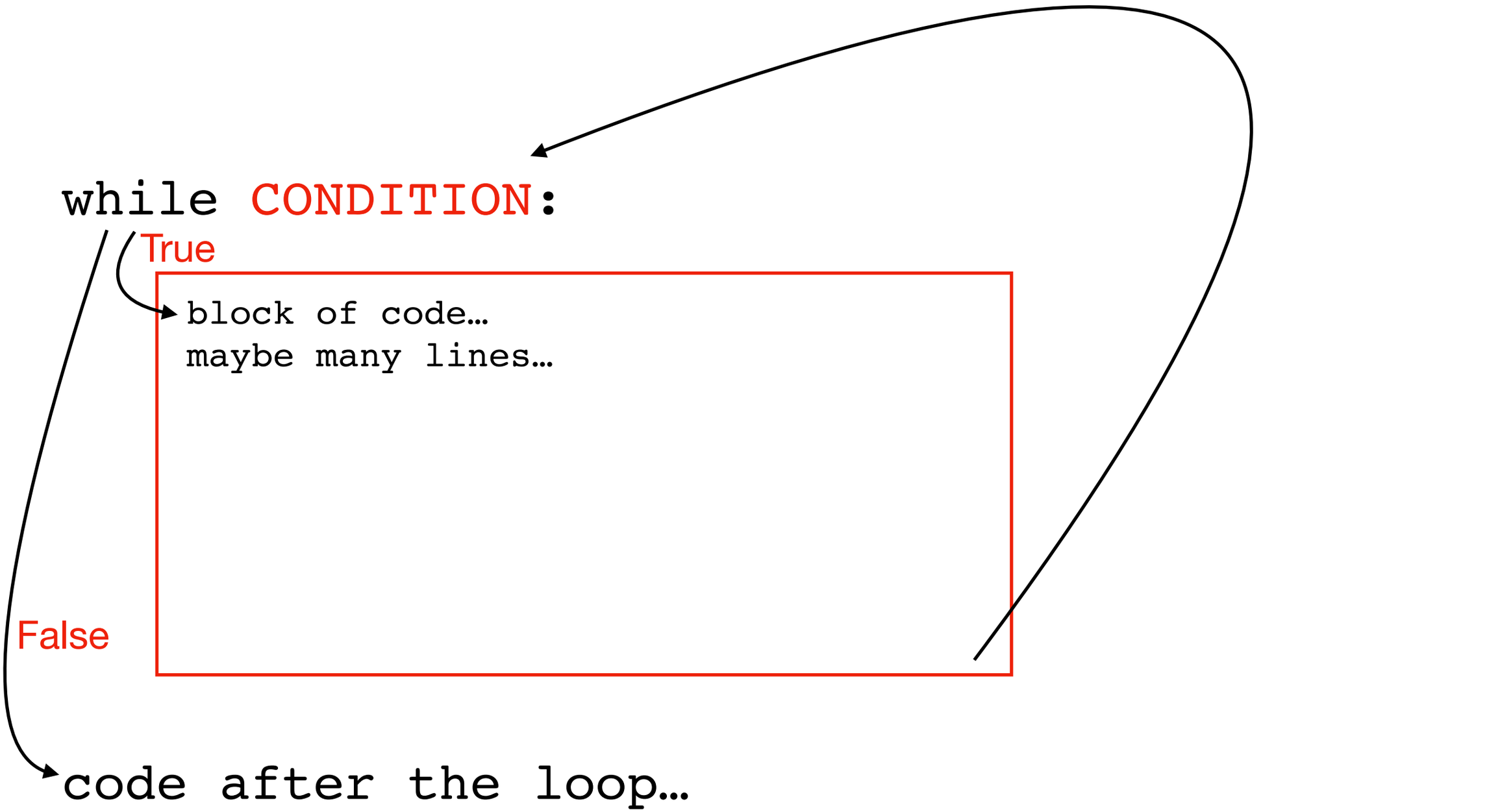
Worksheet

Break

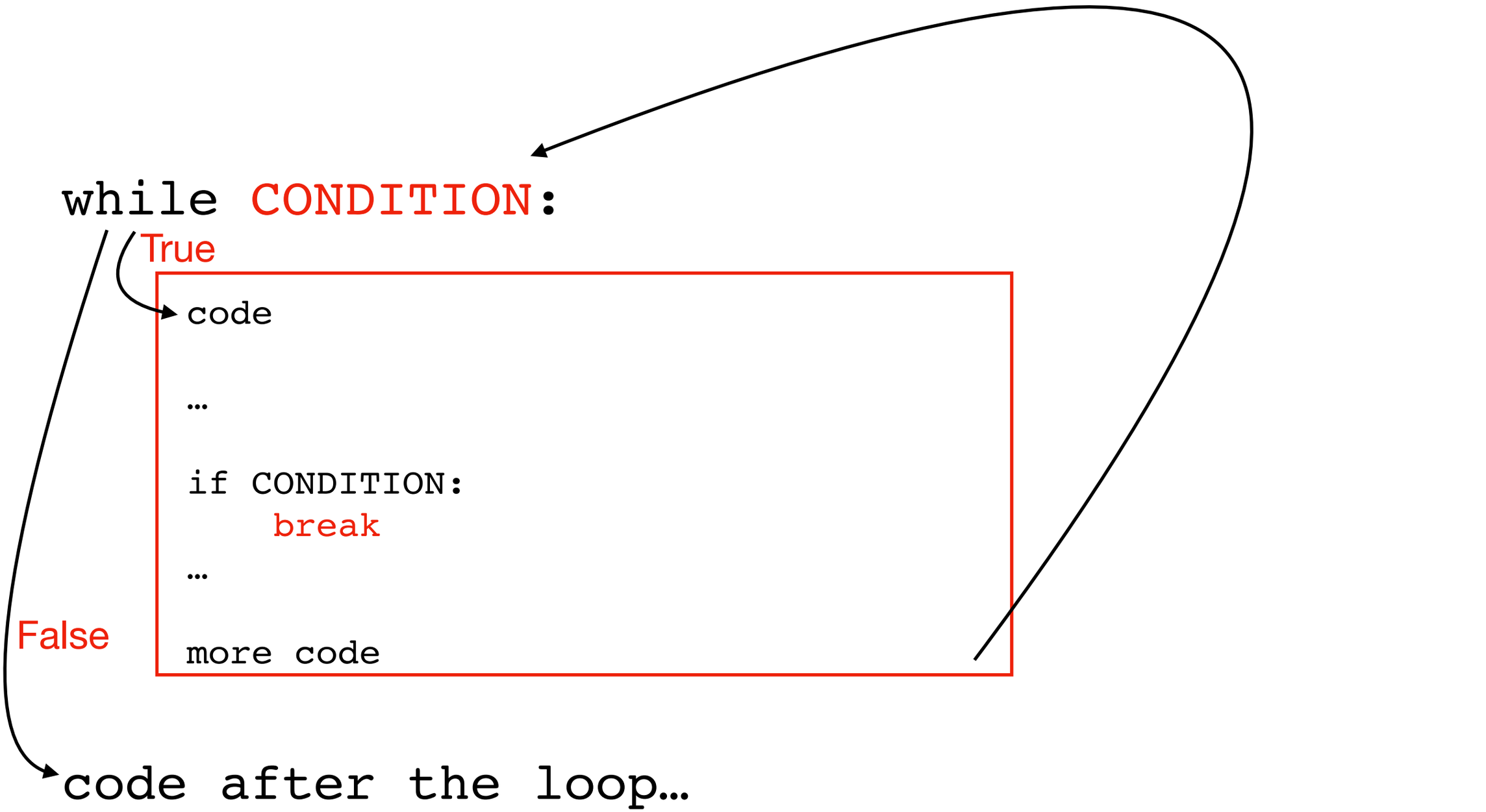
Continue

Nesting

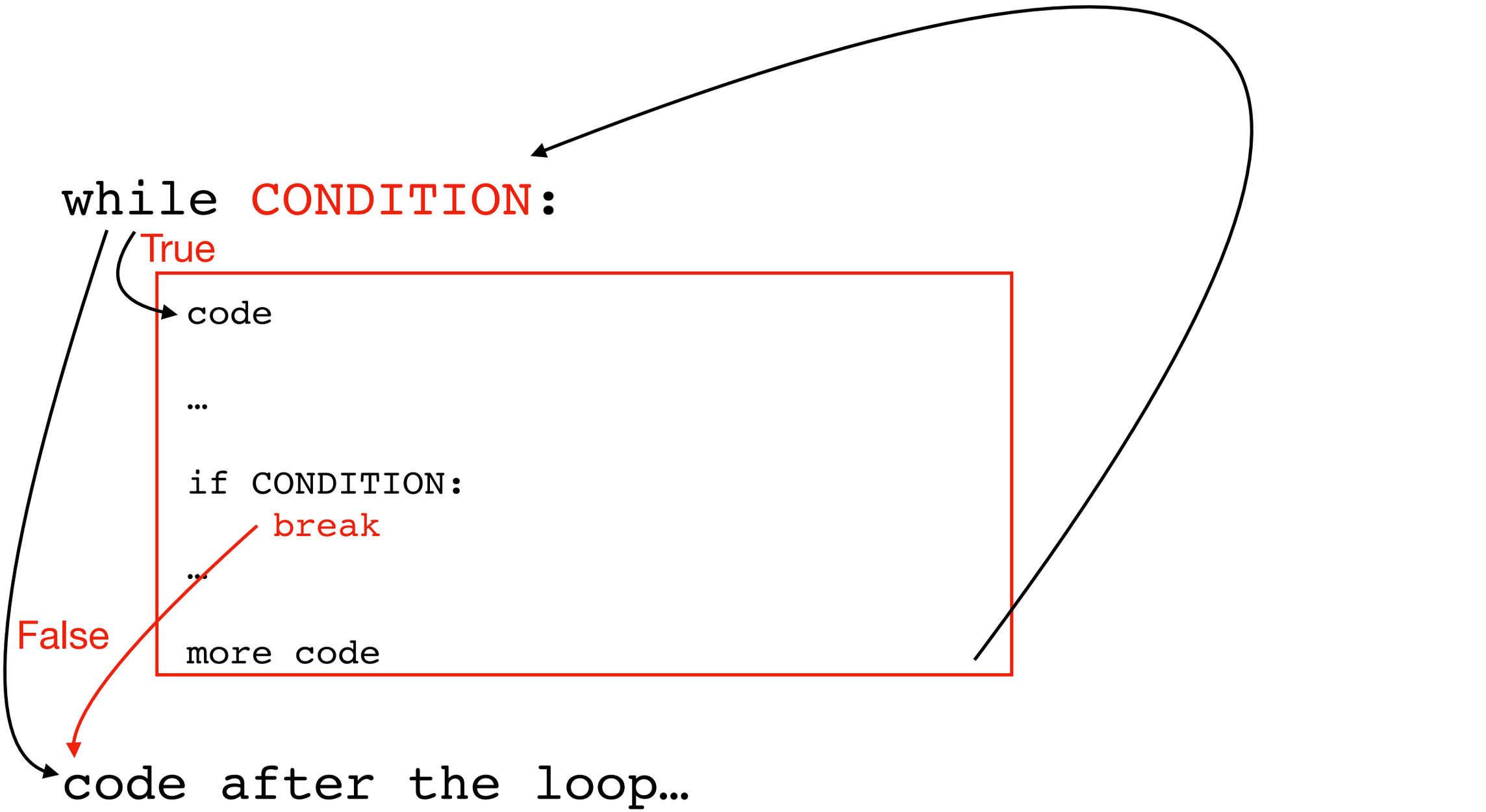
Basic Control Flow



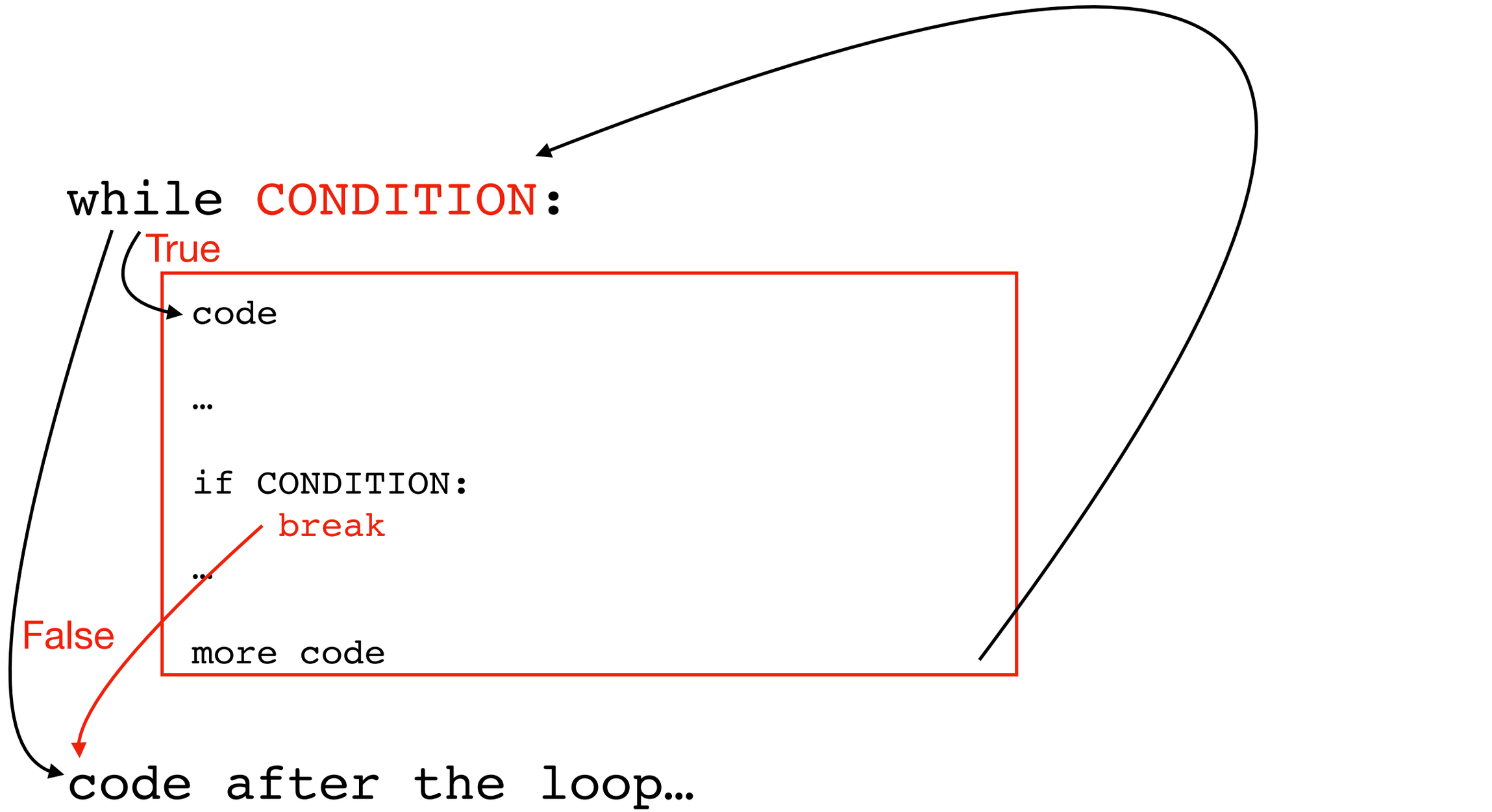
Basic Control Flow



Basic Control Flow

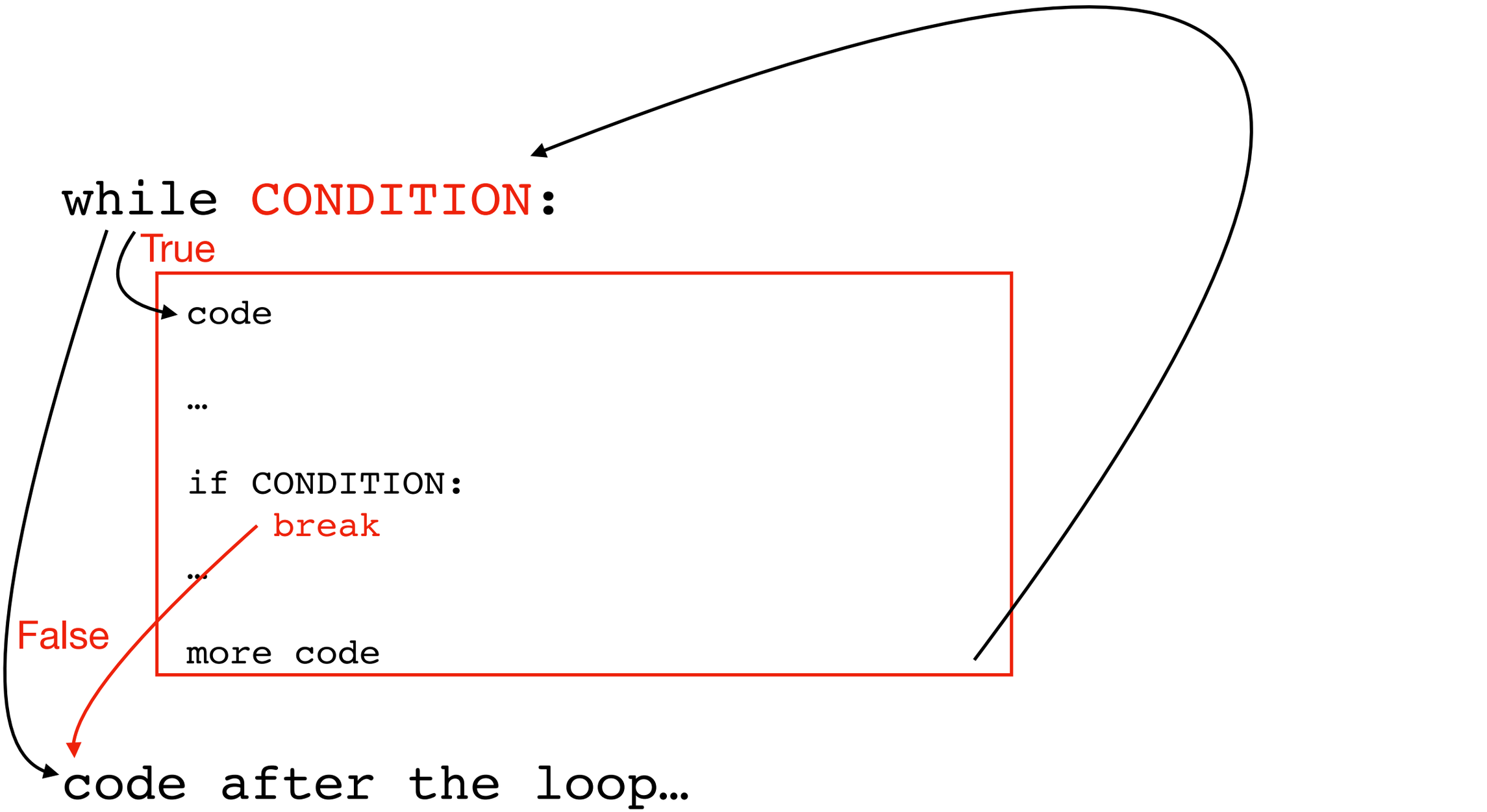


Basic Control Flow



Just like “return” immediately exits a function,
“break” immediately exits a loop

Basic Control Flow



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

Note: this is not a good demo to try coding up yourself during lecture (but maybe after)

Examples:

14 to 16 => NO (because 14, 15, and 16 are all not prime)
20 to 28 => YES (because 23 is prime)

Today's Outline

Design Patterns

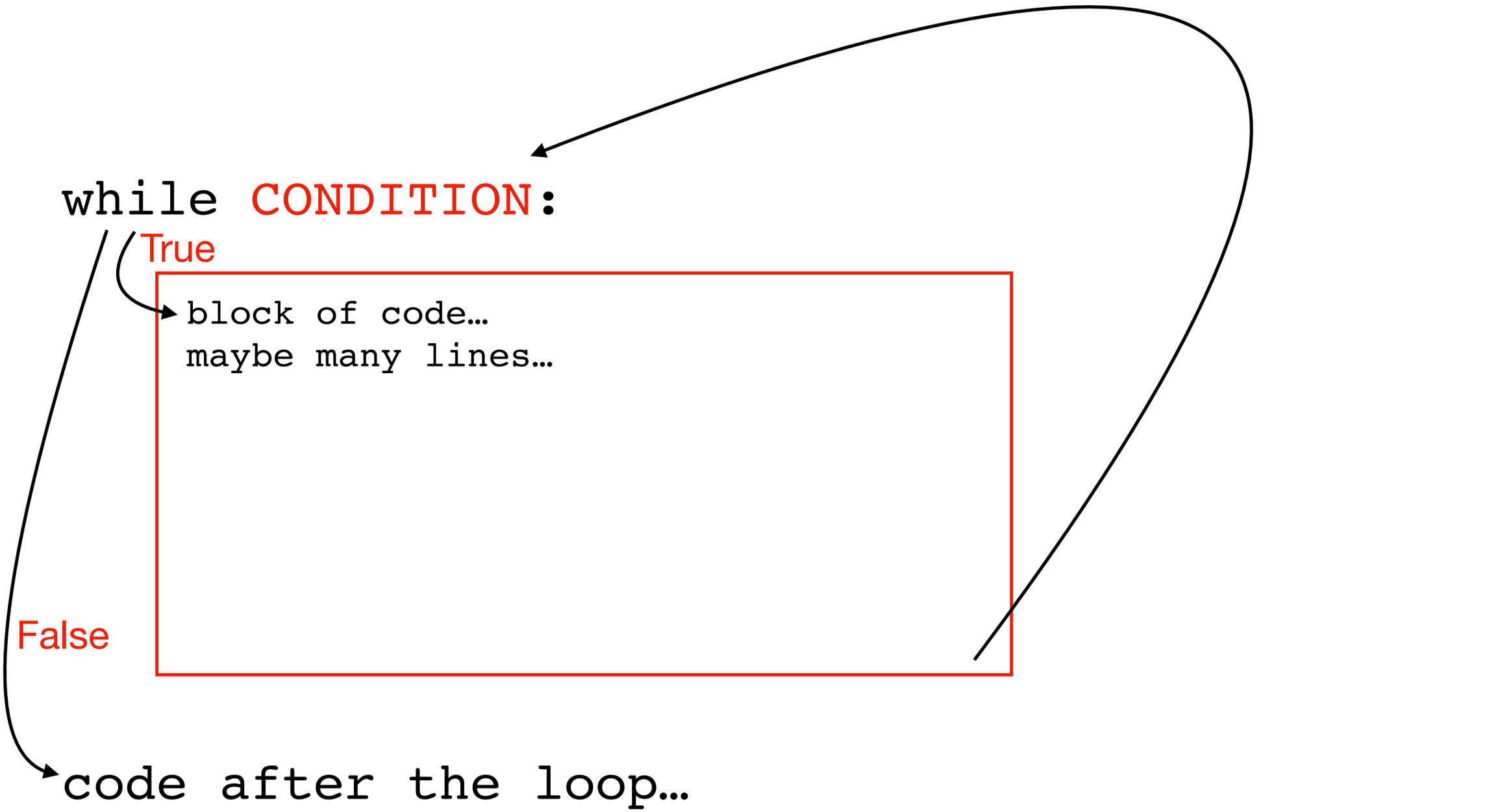
Worksheet

Break

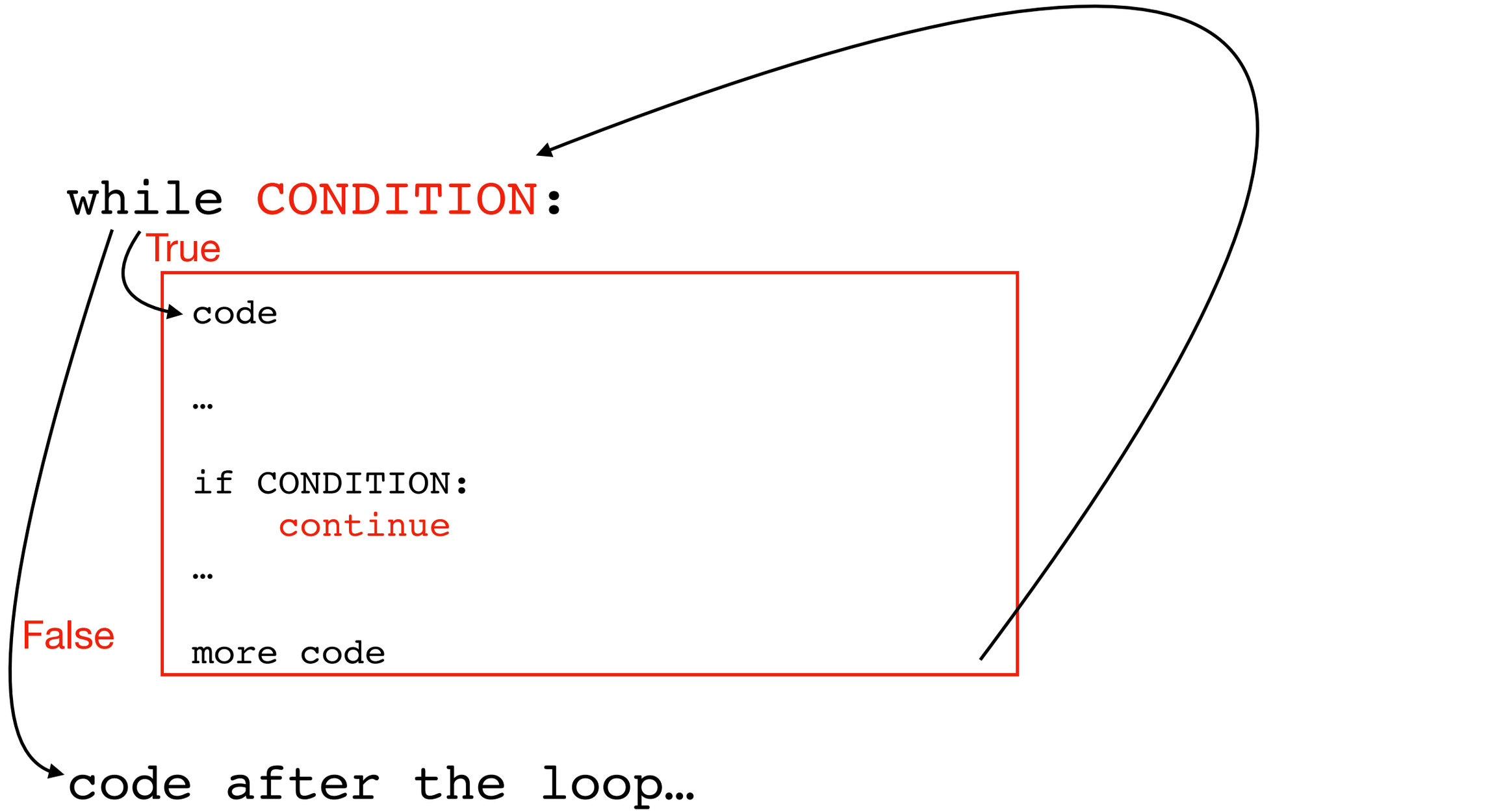
Continue

Nesting

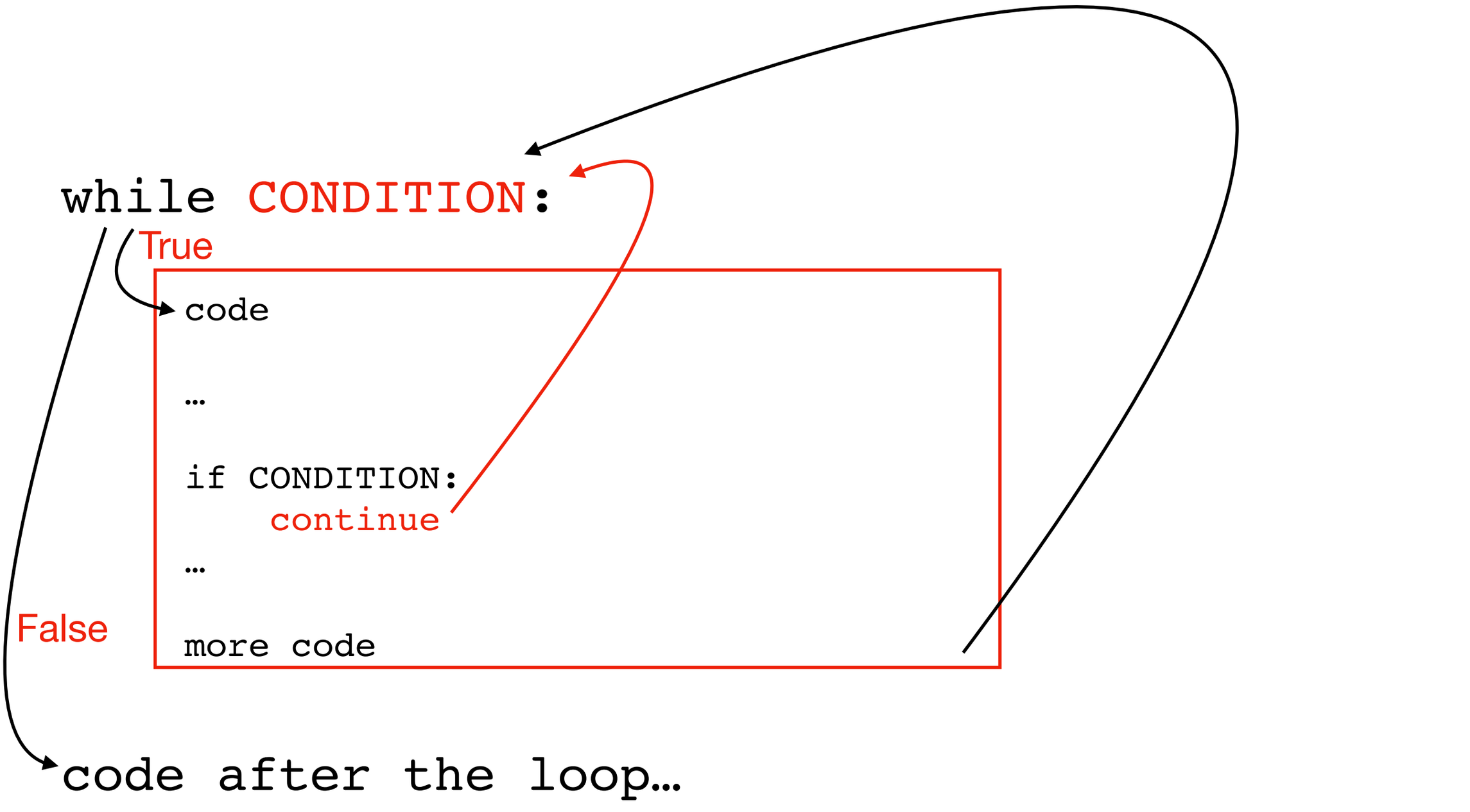
Basic Control Flow



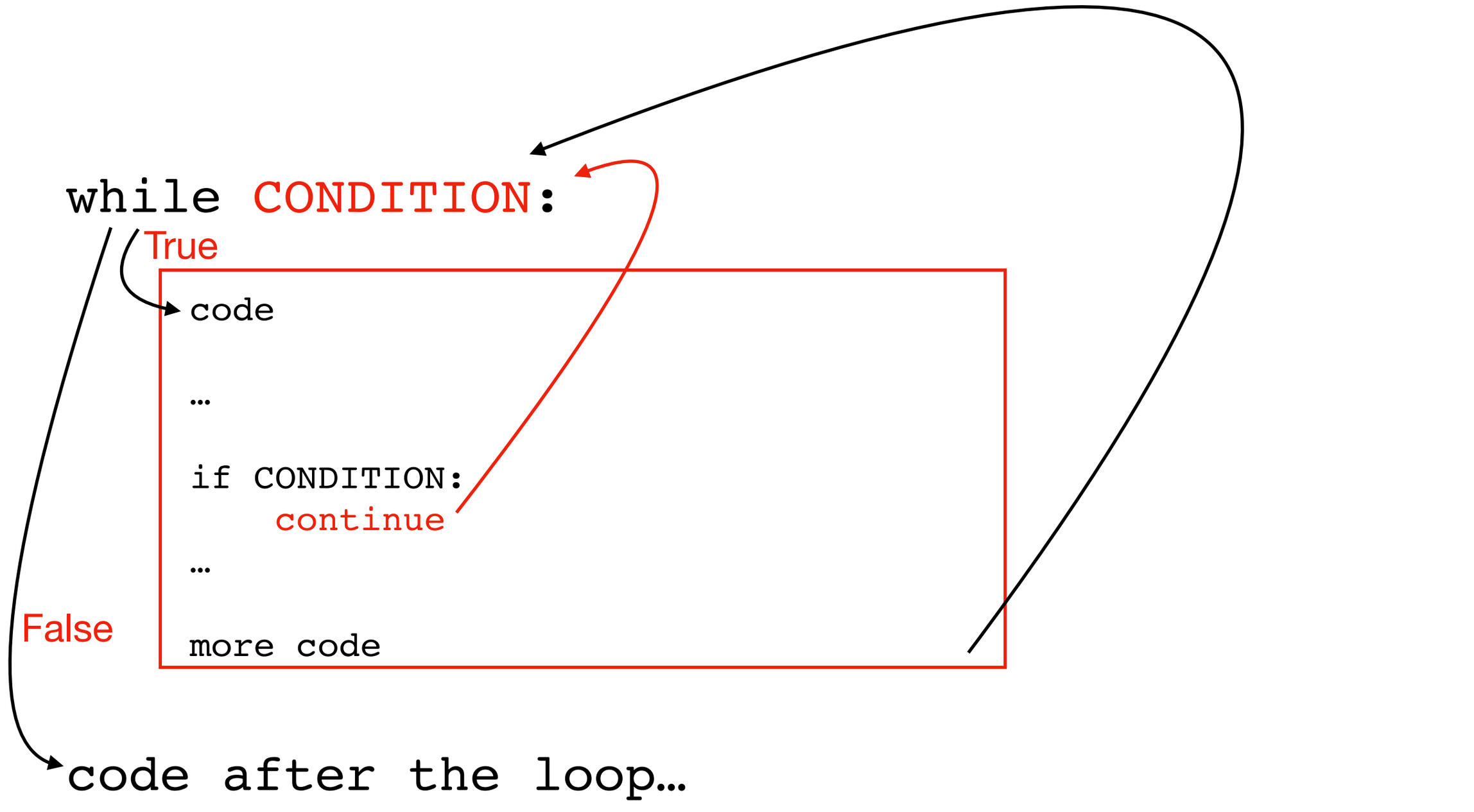
Basic Control Flow



Basic Control Flow



Basic Control Flow



“continue” immediately stops current iteration and goes back to the condition, potentially to start another iteration

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): **70**
avg is 60
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
```

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
```

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