

CSCI 145 · Introduction to Computing · Fall 2026

Iteration

Lecture 6 — repeating code with `while` and `for` loops

Outline

- `while` loops

- `for` loops and `range()`

- Choosing between `for` and `while`

Part 1 of 3

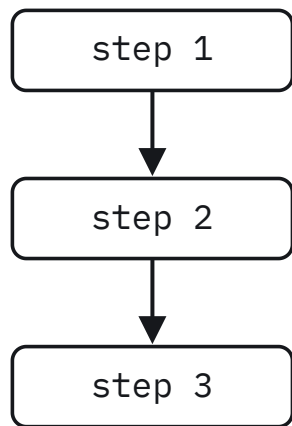
while loops

Control flow

Control flow is the order in which a program's statements run.

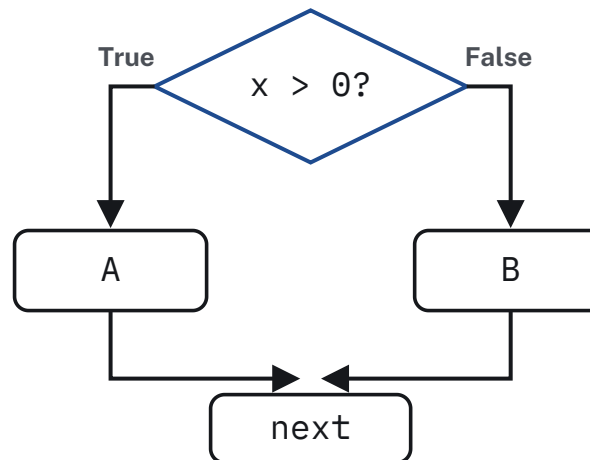
Sequence

top to bottom, one after another



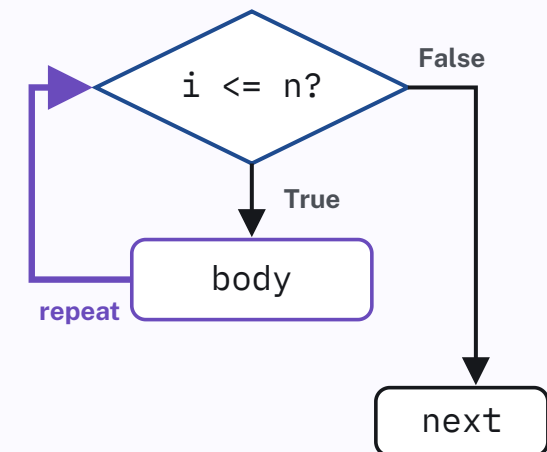
Selection — `if`

choose which path to take



Repetition — loops

take the same path again and again



Why loops? Print all divisors of 9

Without a loop

```
if 9 % 1 == 0:
    print(1)
if 9 % 2 == 0:
    print(2)
if 9 % 3 == 0:
    print(3)
if 9 % 4 == 0:
    print(4)
if 9 % 5 == 0:
    print(5)
```

...9 copies of nearly the same code. What if `n` were 1,000?

With a loop

```
n = 9
i = 1
while i <= n:
    if n % i == 0:
        print(i)
    i += 1
```

← same as `i = i + 1`

Write the repeated part **once**. Works for any `n`.

A `while` loop looks a lot like an `if`

`if` : run the block at most once

```
x = 3
if x > 0:
    print(x)
    x = x - 1
```

Output:

3

`while` : repeat while it's True

```
x = 3
while x > 0:
    print(x)
    x = x - 1
```

Output:

**3
2
1**

Anatomy of a `while` loop

new keyword

`while` starts the loop

condition

Just like an `if`

Any expression that is `True` or `False`

...then repeat

After the body, jump back up
Check the condition again

`while` *condition:*

statement 1

statement 2

...

body

The indented block runs
when the condition is `True`

⚠ make progress

The body must eventually make the condition `False`, or the loop never ends

Tracing a `while` loop

```
n = 9
i = 1
while i <= n:
    if n % i == 0:
        print(i)
    i += 1
```

n 9 **i** 10

LOOP ENDS

Back to the top: `10 <= 9` is False, so the loop stops. Every divisor of 9 is printed.

OUTPUT

1
3
9

Done!

Infinite loops

If the condition never becomes `False`, the loop **never ends**.

✓ ends

```
n = 9
i = 1
while i <= n:
    if n % i == 0:
        print(i)
    i += 1
```

∞ never ends

```
n = 9
i = 1
while i <= n:
    if n % i == 0:
        print(i)
```

no i += 1: i is 1 forever

To stop a runaway program in Thonny, click the  button.

Which of these loops eventually stop?

A

```
i = 0
while i < 20:
    print(i)
    i = i + 2
```

B

```
i = 0
while i < 20:
    print(i)
    i = i + 2
```

C

```
i = 5
while i > 1:
    print(i)
    i = i + 1
```

Go to **slido.com** and enter the code:

3674841

Or [join directly here](#)



Which of these loops eventually stop?

A

```
i = 0
while i < 20:
    print(i)
    i = i + 2
```

✓ A stops

Prints 0, 2, 4, ..., 18. Then `i` is 20, and `20 < 20` is False.

B

```
i = 0
while i < 20:
    print(i)
i = i + 2
```

∞ B never stops

`i = i + 2` isn't indented, so it's **not in the loop**. Prints 0 forever.

C

```
i = 5
while i > 1:
    print(i)
    i = i + 1
```

∞ C never stops

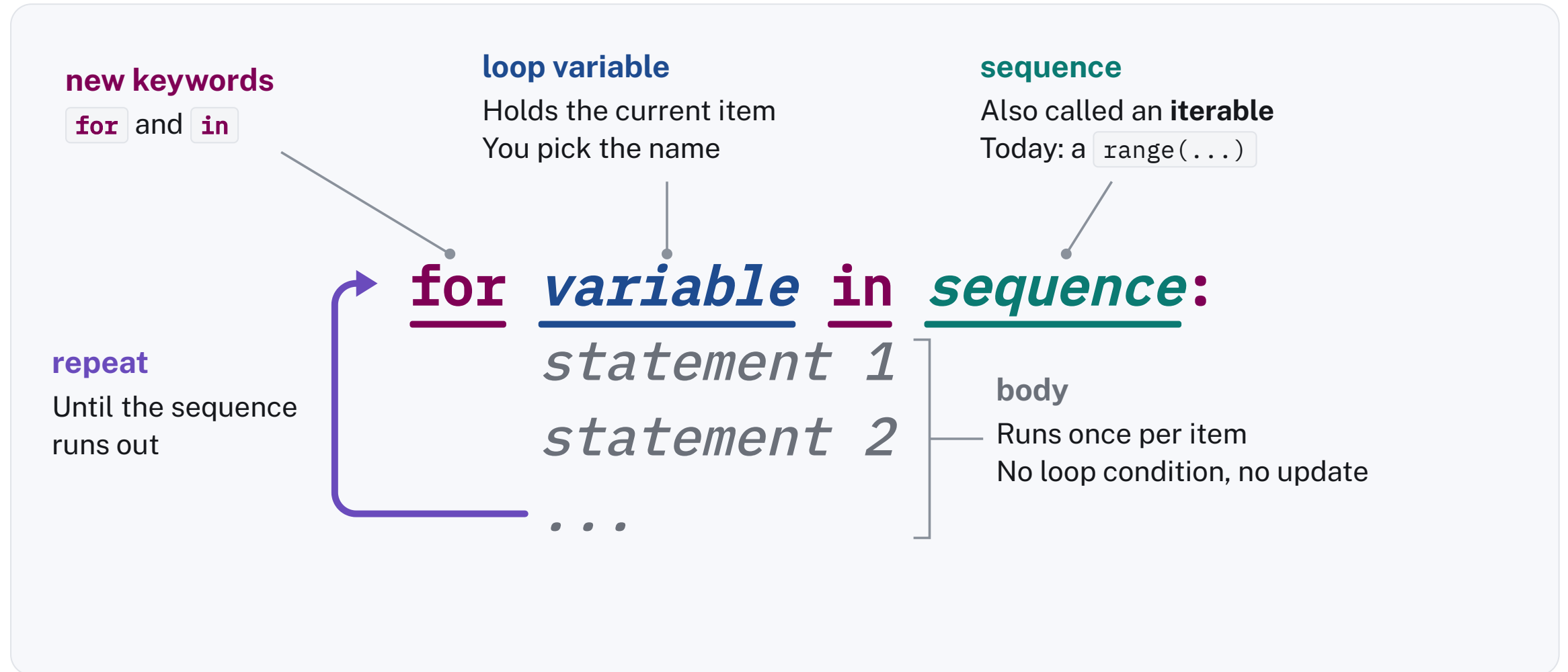
`i` moves the **wrong way**: 5, 6, 7, ... is always > 1 .

Answer: only A. The update must be **inside** the loop and move toward False.

Part 2 of 3

for loops and **range()**

Anatomy of a `for` loop



Counting with `range`

`range(start, stop)`

`range(1, 5)`



start = 1: included

stop = 5: not included

- Whole numbers from `start` up to, but **not including**, `stop`
- Same rule as slicing: recall `s[1:5]` also stops **before** index 5.
- In a `for` loop, the loop variable takes each of these values in turn.

Which values does `i` take?

```
for i in range(1, n + 1):
```

With `n = 9`, that's `range(1, 10)`, which gives `i` these values, in order:

1 2 3 4 5 6 7 8 9 ~~10~~

Why `n + 1`? The stop value is **not** included, so to reach `n` itself (9), stop has to be one past it (10).

Tracing a **for** loop

```
n = 9
for i in range(1, n + 1):
    if n % i == 0:
        print(i)
```

n 9 i 9

VALUES i WILL TAKE FROM range(1, n + 1)

1 2 3 4 5 6 7 8 9 ~~10~~

LOOP ENDS

No values left in the range, so the loop ends. Note **i** never becomes 10: the stop value is not included.

OUTPUT

1
3
9

Done!

The `range` function

Three ways to call it. The stop value is **never** included.

`range(5)` →

0	1	2	3	4	5
---	---	---	---	---	--------------

just a stop: starts at 0

`range(2, 6)` →

2	3	4	5	6
---	---	---	---	--------------

start and stop: counts up by 1

`range(0, 10, 2)` →

0	2	4	6	8	10
---	---	---	---	---	---------------

start, stop, and **step**: counts up by 2

What does this print?

```
for i in range(3, 12, 3):  
    print(i)
```

Go to **slido.com** and enter the code: 3674841

Or [join directly here](#)



What does this print?

```
for i in range(3, 12, 3):  
    print(i)
```

Start at 3, count up by 3, stop **before** 12:

3 6 9 ~~12~~

Output:

3
6
9

12 is the stop value, so it is not printed.

Part 3 of 3

Choosing between `for` and `while`

for loops can be written as **while** loops

What goes in the gray boxes so the two loops print the same thing?

for

```
for i in range(0, 20):  
    print(i)
```

while

```
i = 0  
while i < 20:  
    print(i)  
    i = i + 1
```

start → set `i` up before the loop

stop → the condition

step (1 by default) → the update

A **for** loop does all three for you, so there's no update to forget.

Exercise: `for` or `while` ?

Which kind of loop fits each task best? (No code, just decide.)

Print the numbers from 1 to 1000

for
(while works too)

Keep asking for a password until the user types the right one

while
no way to know how
many tries

Print every character of a string, one per line

for
one per index:
`range(len(s))`

RULE OF THUMB

- `for` when you know **how many times** (or have a sequence to traverse)
- `while` when you repeat **until something happens**

Coming next time: nested loops

PREVIEW

- A loop's body can contain **another** loop
- The inner loop runs all the way through for **each** pass of the outer loop

```
for i in range(1, 6):  
    for j in range(1, 6):  
        print(i * j, end=" ")  
    print()    # end the row
```

i = 1	1	2	3	4	5
i = 2	2	4	6	8	10
i = 3	3	6	9	12	15
i = 4	4	8	12	16	20
i = 5	5	10	15	20	25

Activity

1. Download [day06_iteration.py](#) and save it in your `cs145/class` folder.
2. Launch Thonny. From the File menu, open `cs145/class/day06_iteration.py`.
3. For part 1, predict what each loop prints, then uncomment it and run the file to check.
4. For part 2, complete `countdown(n)` with a `while` loop and `print_multiples(k, limit)` with a `for` loop and `range`.
5. If you have time, complete `print_powers_of_two(limit)`. Which kind of loop fits?

To do

- Review this slide deck and the corresponding notes: [Notes 7](#)
 - Exam 1 is this Friday, covering Lectures 1–4
 - Quiz 3 (Lectures 5–6) is at the start of next class
- See [practice materials](#)

Next time

Lecture 7 — more iteration: looping over strings, accumulators, nested loops