

CSC 141 / FALL 2026

Loops: while, for, and range

CSC 141 · Class 14

Thursday, October 15, 2026

Dr. Si Chen

Where we are

Classes 1–11: values, decisions, and functions.

Midterm 1 covered all of that — no loops.

Today: repetition — making the computer do a step many times.

Everything you know still applies: a loop body is just ordinary code.

Today's learning goals

Write a while loop with a condition and an update.

Trace a loop by hand with a table.

Use for and range() to repeat a known number of times.

Build totals and counts with the accumulator pattern.

Why loops?

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
# ...and what if we need 1000?
```

Fine for five lines.

Not for 1000 — or for "until the input is valid."

A loop repeats a block for us.

The while loop

```
count = 1
while count <= 5:
    print(count)
    count = count + 1
print("done")
```

Condition
checked before
every pass.

True: run the
body, check
again.

False: go on
after the loop.

Trace the while loop

count	count <= 5	printed
1	True	1
2	True	2
3	True	3
4	True	4
5	True	5
6	False	(loop ends)

Bug: the loop that never ends

```
count = 1
while count <= 5:
    print(count)
# forgot: count = count + 1
```

count stays 1:
True forever.

Infinite loop —
stop it with
Ctrl+C.

Need an update
that moves
toward False.

while: repeat until the input is valid

```
age = int(input("Age: "))
while age < 0:
    print("Age cannot be negative.")
    age = int(input("Age: "))
print("Thanks, you are", age)
```

Unknown
number of tries.

The input() inside
the loop is the
update.

Try: -3, -1, 19

The for loop with range()

```
for i in range(5):  
    print("Pass", i)
```

range(5) gives 0,
1, 2, 3, 4.

i takes each
value in turn.

No manual
update needed.

range(start, stop, step)

call	values produced
range(5)	0 1 2 3 4
range(1, 6)	1 2 3 4 5
range(0, 10, 2)	0 2 4 6 8
range(5, 0, -1)	5 4 3 2 1
range(3, 3)	(nothing)

Same loop, two ways

```
# while: we manage the counter
count = 1
while count <= 5:
    print(count)
    count += 1
```

```
# for: range manages it
for count in range(1, 6):
    print(count)
```

Both print 1 to 5.

Known count:
for.

Until something
happens: while.

`+= 1` means
"add 1".

The accumulator pattern

```
total = 0
for n in range(1, 6):
    total = total + n
print(total)
```

1. Start the total before the loop.

2. Add to it inside the loop.

3. Use it after the loop.

Output: 15

Trace the accumulator

n	total before	total after
1	0	1
2	1	3
3	3	6
4	6	10
5	10	15
	loop ends	print 15

The counting pattern

```
count = 0
for n in range(1, 21):
    if n % 3 == 0:
        count += 1
print(count, "multiples of 3")
```

Add 1 only when
the if is True.

Decisions inside
a loop.

Output: 6
multiples of 3

Loops inside functions

```
def sum_to(n):  
    total = 0  
    for k in range(1, n + 1):  
        total += k  
    return total
```

```
print(sum_to(5))  
print(sum_to(100))  
print(sum_to(0))
```

$n + 1$ so n itself
is included.

Output: 15,
5050, 0

Boundary test: n
 $= 0$.

Common loop bugs

Off by one: `range(1, n)` stops at $n - 1$. Use `range(1, n + 1)` to include n .

Reset inside the loop: `total = 0` belongs before the loop, not in it.

`return` inside the loop: the function returns after the first pass.

No update in a while loop: the condition never becomes `False`.

Guided build: savings goal

Write `weeks_to_goal(goal, deposit)`.

Start with `saved = 0` and `weeks = 0`.

Each week, add `deposit` to `saved` and add 1 to `weeks`.

Stop as soon as `saved` reaches the goal; return `weeks`.

Which loop fits — `while` or `for`? Why?

Savings goal test cases

goal	deposit	expected weeks
100	25	4
100	30	4
0	10	0
50	50	1

Savings goal solution

```
def weeks_to_goal(goal, deposit):  
    saved = 0  
    weeks = 0  
    while saved < goal:  
        saved += deposit  
        weeks += 1  
    return weeks  
  
print(weeks_to_goal(100, 30))
```

Condition: keep going while we are short of the goal.

Update: saved grows every pass.

Output: 4

Independent practice

Use for and range() in loop_practice_starter.py.

sum_of_squares(n): $1^2 + 2^2 + \dots + n^2$

sum_of_squares(3) is 14, sum_of_squares(0) is 0

count_divisible(n, d): how many of 1..n divide by d

count_divisible(20, 3) is 6, count_divisible(10, 5) is 2

Exit ticket

What values does `range(2, 8, 2)` produce?

What happens if a while loop never changes its condition variable?

In the accumulator pattern, where does `total = 0` go, and why?

Before next class

Finish the independent practice and run every test.

Try it in the Python Playground:

wcu.ninja/csc141/python

Next class (Tue Oct 20): more loops — nested loops and looping over strings.