Loops: for and while

Loops repeat work without rewriting it — learn for loops with range, while loops, break and continue, and how to avoid infinite loops.

Introduction

Imagine printing the numbers 1 to 100 by writing 100 `print()` lines. That would be long, boring, and full of typing mistakes. A **loop** solves this: you write the work once, and Python repeats it as many times as you say.

Python has two main loops. A **for loop** repeats over a known group — a list, a string, or a range of numbers. A **while loop** repeats while a condition stays True — for example, while the user has not typed the right answer yet. Both use an indented block, just like `if` does.

Loops are the engine of most programs: showing every item in a list, counting, adding up totals, and repeating a game round. In this lesson you will use both kinds, stop a loop early with `break`, skip one round with `continue`, and learn the one danger every beginner must respect — the infinite loop.

(Roman Urdu: Loop ka matlab hai aik kaam ko baar baar karna, baghair usay baar baar likhay — for loop aik group par chalta hai, while loop tab tak chalta hai jab tak shart poori ho.)

The loop itself is rarely the hard part — choosing the right kind is. If you know how many rounds you need, a `for` loop with `range()` is almost always clearer and safer than a `while` loop. Reach for `while` when the number of rounds is unknown: waiting for the right answer, reading until the file ends, or repeating a game round until someone wins.

Key points

- **A for loop visits each item in a group.** `for name in names:` runs the block once per name.
- **`range()` makes a sequence of numbers.** `range(5)` gives 0, 1, 2, 3, 4. `range(1, 6)` gives 1 to 5. `range(1, 11, 2)` gives odd numbers 1, 3, 5, 7, 9.
- **A while loop repeats while its condition is True.** It checks the condition before every round, so it can run zero times.
- **The loop variable must change in a while loop.** If nothing inside the loop changes the condition, the loop never ends.
- **`break` exits the loop immediately.** Python jumps to the first line after the loop.
- **`continue` skips the rest of the current round.** Python jumps straight to the next round.
- **A nested loop is a loop inside a loop.** The inner loop runs fully for every single round of the outer loop.
- **An infinite loop never ends.** It happens when a while condition never becomes False. In a runner, stop the program and fix the condition or the update step.

- **Indentation defines the loop body.** Only indented lines repeat; the first unindented line runs once, after the loop ends.
- **`enumerate()` gives you the position as well as the item.** `for index, name in enumerate(names):` counts from 0 automatically, so you do not need a separate counter.
- **A `for...else` block runs its `else` only if no `break` happened.** It is the tidy way to say "we checked everything and found nothing".

Examples

Example 1 — A for loop with range

```
for number in range(1, 6):
    print(number)

total = 0
for number in range(1, 11):
    total = total + number
print(f"Sum of 1 to 10: {total}")
```

What you will get:

```
1
2
3
4
5
Sum of 1 to 10: 55
```

`range(1, 6)` stops before 6, so it gives 1 to 5. The second loop adds every number from 1 to 10 into `total`.

Example 2 — A while loop with break and continue

```
count = 0
while count < 10:
    count = count + 1
    if count == 3:
        continue
    if count == 7:
        break
    print(count)
print("Loop finished")
```

What you will get:

```
1
2
4
5
6
Loop finished
```

Number 3 is skipped by `continue`, and the loop stops completely at 7 because of `break`. The key line is `count = count + 1` at the top — without it, `count` would stay 0 forever and the loop would never end.

Example 3 — A nested loop: a small multiplication table

```
for row in range(1, 4):
    for column in range(1, 4):
        print(f"{row} x {column} = {row * column}")
    print("---")
```

What you will get:

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
---
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
---
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
---
```

For each value of `row`, the inner loop runs all three columns before the outer loop moves on. That is why the output has three groups of three lines.

Example 4 — Looping over a string, and counting with enumerate

A `for` loop works on any group — including a string, where each round gives you one letter. When you also need the position, `enumerate()` supplies it:

```
word = "Python"
for letter in word:
    print(letter)
for index, name in enumerate(["Ali", "Ayesha", "Bilal"]):
    print(index, name)
```

What you will get:

```
P
y
t
h
o
n
0 Ali
1 Ayesha
2 Bilal
```

Without `enumerate()`, beginners usually create a counter variable by hand and forget to update it. Let Python count for you.

Example 5 — Searching a list, with for...else

You need to know whether any mark is a fail. Loop through, break the moment you find one — and if the loop finishes without breaking, the `else` reports that everyone passed:

```
marks = [85, 90, 78, 92]
for mark in marks:
    if mark < 50:
        print("Found a fail:", mark)
        break
else:
    print("No fail marks — everyone passed")
```

What you will get:

```
No fail marks — everyone passed
```

Change 78 to 40 and run it again: the loop breaks at 40, prints "Found a fail: 40", and the `else` is skipped. That skip is the whole point of `for...else` — it runs only on a clean finish.

Example 6 — Real-world case: a shop's weekly sales summary

A shopkeeper records each day's sales in a list. One loop adds them up, and the average follows from the total and the number of days:

```
daily_sales = [1200, 800, 1500, 900]
total = 0
for sale in daily_sales:
    total = total + sale
print("Week total:", total)
print("Average per day:", total / len(daily_sales))
```

What you will get:

```
Week total: 4400
Average per day: 1100.0
```

1200 + 800 + 1500 + 900 = 4400, and 4400 / 4 = 1100. Add a fifth day to the list and both numbers update — no extra code needed. That is the payoff of loops: the same code handles 4 days or 400.

Quick recap

- `for` loops over a group or a `range()` of numbers.
- `while` loops while a condition is True — and the condition must be able to become False.
- `break` exits the loop; `continue` skips to the next round.
- A nested loop runs the inner loop fully for each outer round.
- An infinite loop means the while condition never turns False — stop and fix it.
- `enumerate()` gives the index and the item together.
- `for...else` runs its `else` only when the loop was not stopped by `break`.

Common mistakes

1. **Forgetting to update the while variable** — `count` never changes, so the loop never ends. Fix: make sure something inside the loop changes the condition, like `count = count + 1`.
2. **Thinking `range(5)` includes 5** — it gives 0 to 4 only. Fix: to include 5, write `range(6)`, or `range(1, 6)` for 1 to 5.
3. **Forgetting the indent** — unindented lines do not repeat. Fix: indent every line that belongs in the loop body.
4. **Using `break` when you meant `continue`** — `break` ends the whole loop; `continue` only skips one round. Fix: ask yourself "stop everything, or skip just this one?"
5. **Writing a while loop where a for loop fits** — counting a fixed number of rounds is simpler with `for` and `range()`. Fix: use `for` when you know how many rounds you need.

1. **Changing a list while looping over it** — removing items inside a `for` loop over the same list skips items, because the list shifts under the loop. Fix: loop over a copy, like `for item in items[:]:`, or build a new list.

2. **An off-by-one error in a `while` condition** — `while count <= 5:` runs 6 times when counting from 0, not 5. Fix: trace the first and last values by hand before running.

Practice

1. Print the numbers 1 to 20 with a for loop and `range()`. Then print only the even numbers from 2 to 20.
2. Write a while loop that counts down from 5 to 1 and then prints "Go!". Check that it stops.
3. Make a list of five names. Loop over it and print each name, but skip one name with `continue` and stop the loop early with `break` on another name.
4. Write a nested loop that prints a 3 by 3 grid of stars, one row per line, using `print("*", end=" ")` inside the inner loop and a plain `print()` after it. Run it and check the shape.
5. Print the 5 times table (5 × 1 up to 5 × 10) with a `for` loop and `range()`. Then do the same for 7.
6. Count the vowels in a word with a loop: loop over each letter, and add 1 to a counter when the letter is in `"aeiou"`. Test it with `"education"` (answer: 5).
7. Write a `while` loop that keeps adding numbers 1, 2, 3, ... to a total until the total passes 100. Print how many numbers it took, and the final total.

Practice quiz

1. What values does range(3) give?

- 1, 2, 3
- 0, 1, 2
- 3, 2, 1
- 0, 1, 2, 3

Answer: 0, 1, 2

2. What does break do inside a loop?

- Skips one step
- Stops the loop immediately
- Restarts the loop
- Pauses for one second

Answer: Stops the loop immediately

3. What is the danger of a while loop?

- It runs too fast
- If its condition never becomes false, it runs forever
- It cannot use numbers
- It deletes variables

Answer: If its condition never becomes false, it runs forever

---

Watch the video lesson: https://youtu.be/usBgMxNZ50Q

Read online: https://scopetreasure.com/notes/py-loops · More free tools and notes at https://scopetreasure.com