

Iteration

Python Reference

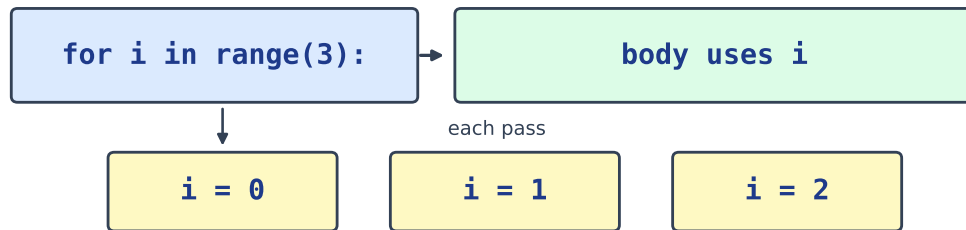
for loops and range

A loop repeats code. A for loop repeats once for each item in a sequence . `range(n)` gives the numbers 0 up to `n - 1`.

```
for i in range(5):  
    print(i)  
# 0, then 1, 2, 3, 4 (each on its own line)
```

- `range(a, b)` goes from `a` up to (but not including) `b`.
- `range(a, b, step)` adds a step each time.

```
for n in range(2, 11, 2):  
    print(n)          # 2 4 6 8 10
```



Exam tip: `range(n)` is 0 ... `n-1`; the loop variable takes one value per pass

for `i` in `range(n)`: body runs with `i` = 0 ... `n-1`

The accumulator pattern

To build a result across a loop, start a variable **before** the loop, then update it each turn. This is the accumulator pattern.

```
total = 0  
for n in range(1, 6):  
    total = total + n  
print(total)          # 15
```

- The same idea counts how many items match a test.

```
count = 0  
for letter in "banana":  
    if letter == "a":  
        count = count + 1  
print(count)          # 3
```

while loops

A while loop repeats **as long as** a condition stays true. Change something inside, or it never stops — an infinite loop .

```
n = 1  
while n <= 3:  
    print(n)  
    n = n + 1  
# 1 2 3
```

- `break` leaves the loop straight away.

```
total = 0  
while True:  
    total = total + 10  
    if total >= 30:  
        break  
print(total)          # 30
```

Nested loops

A loop inside another loop is a nested loop . The inner loop runs fully for **each** turn of the outer loop .

```
for row in range(3):
    line = ""
    for col in range(3):
        line = line + "*"
    print(line)
# ***
# ***
# ***
```

Common mistakes

- `range(n)` goes from 0 to `n - 1`, not 1 to `n` — the classic off-by-one error.
- Changing a list while looping over it skips items; loop over a copy if you must remove things.
- Forgetting to change the loop variable in a **while** makes it run forever.
- Indentation decides what is inside the loop; a wrongly-indented line runs only once, after the loop.