

A loop runs the same lines again: once for each item (for), or as long as a condition is true (while). Examples use `prices = [4.5, 12.0, 3.25]` and `names = ["Ada", "Grace", "Linus"]`.

FOR: ONE ITEM AT A TIME

```
for p in prices:
    print(p)
```

p takes each value in turn: 4.5, then 12.0, then 3.25

```
out = []
for c in "Ada":
    out.append(c)
```

out
→ ['A', 'd', 'a']
a loop over a string gives one character at a time

```
out = []
for k in {"tea": 3, "pie": 4}:
    out.append(k)
```

out
→ ['tea', 'pie']
a loop over a dict gives its keys

RANGE: A RUN OF NUMBERS

```
list(range(5))
```

→ [0, 1, 2, 3, 4]
starts at 0 and stops before 5

```
list(range(2, 6))
```

→ [2, 3, 4, 5]
range(start, stop): stop is not included

```
list(range(0, 10, 3))
```

→ [0, 3, 6, 9]
the third number is the step: count up by 3

```
list(range(5, 0, -1))
```

→ [5, 4, 3, 2, 1]
a negative step counts down

ENUMERATE AND ZIP

```
list(enumerate(names, start=1))
```

→ [(1, 'Ada'), (2, 'Grace'), (3, 'Linus')]
gives a count with each item; start=1 starts the count at 1

```
for i, name in enumerate(names, start=1):
    print(i, name)
```

the usual loop when you need the position

```
list(zip(names, [90, 85, 77]))
```

→ [('Ada', 90), ('Grace', 85), ('Linus', 77)]
pairs items from two lists

```
for name, score in zip(names, scores):
    print(name, score)
```

goes through two lists side by side: each name with its score. zip stops at the end of the shorter list

WHILE: REPEAT WHILE TRUE

```
money, years = 1000, 0
while money < 2000:
    money = money * 1.1
    years += 1
```

years
→ 8
the condition is checked before each pass (each run of the loop body). Use while when you do not know how many passes you need

```
while True:
    answer = input("Price (q to quit): ")
    if answer == "q":
        break
    print(float(answer) * 2)
```

while True never stops on its own: break ends the loop

BREAK AND CONTINUE

```
temps = [18, -3, 35, 19]
```

```
hot = None
for t in temps:
    if t > 30:
        hot = t
        break
```

hot
→ 35
break stops the loop at once

```
ok = []
for t in temps:
    if t < 0:
        continue
    ok.append(t)
```

ok
→ [18, 35, 19]
continue skips the rest of the current pass and goes to the next item

ELSE ON A LOOP

The else block runs when the loop ends without break. Read it as "if no break". It works the same on a while loop.

```
for n in [3, 5, 7]:
    if n % 2 == 0:
        break
else:
    n = None
```

n
→ None
there is no even number, so break never runs and the else block runs. With [3, 4, 7], the loop breaks at 4 and else is skipped

NESTED LOOPS

```
out = []
for size in "SM":
    for colour in ["red", "blue"]:
        out.append(size + "-" + colour)
```

out
→ ['S-red', 'S-blue', 'M-red', 'M-blue']
the inner loop runs fully for each pass of the outer loop. A break in the inner loop leaves only the inner loop

LOOP OVER A DICT

```
stock = {"tea": 3, "pie": 0, "cake": 5}
```

```
empty = []
for item, qty in stock.items():
    if qty == 0:
        empty.append(item)
```

empty
→ ['pie']
items() gives each key and value together

ORDER: REVERSED AND SORTED

```
list(reversed(names))
```

→ ['Linus', 'Grace', 'Ada']
the items from last to first; names itself does not change

```
sorted(prices)
```

→ [3.25, 4.5, 12.0]
a new sorted list, so for p in sorted(prices) goes from smallest to largest

BUILD A RESULT

```
total = 0
for p in prices:
    total += p
```

total
→ 19.75
start at 0, then add each item

```
sum(prices)
```

→ 19.75
the built-in sum() does the same

```
big = prices[0]
for p in prices:
    if p > big:
        big = p
```

big
→ 12.0
finds the largest price by hand; max(prices) does the same

```
count = 0
for p in prices:
    if p > 4:
        count += 1
```

count
→ 2
counts the prices over 4

THE SHORT FORM: A COMPREHENSION

```
[p * 2 for p in prices]
```

→ [9.0, 24.0, 6.5]
a loop that builds a list, in one line

```
[p for p in prices if p < 5]
```

→ [4.5, 3.25]
keeps only the prices under 5. More on the comprehensions sheet

WATCH OUT

```
list(range(1, 5))
```

→ [1, 2, 3, 4]
5 is not included, so the list is one number short. For 1 to 5, use range(1, 6)

```
nums = [1, 2, 2, 3]
for n in nums:
    if n == 2:
        nums.remove(n)
```

nums
→ [1, 2, 3]
changing a list while you loop over it skips items. Loop over a copy: for n in nums[:]

```
n = 0
while n < 5:
    print(n)
```

this loop never ends because n never changes. Add n += 1 inside the loop

```
for i in range(len(names)):
    print(i, names[i])
```

range(len(names)) works, but enumerate(names) is clearer

```
for name in names:
    pass
name
```

→ 'Linus'
the loop variable still exists after the loop: it holds the last item