Lists & 2-D lists

Python Reference

Lists

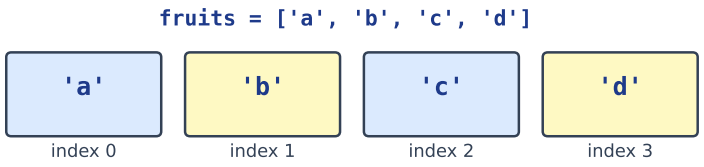
A list 列表 holds many values in order, inside `[]`. Each item 元素 has an index (from 0).

```
scores = [88, 71, 95]
print(scores[0])      # 88
print(len(scores))    # 3
scores.append(60)     # add to the end
print(scores)         # [88, 71, 95, 60]
```

- Change an item by its index: `scores[1] = 100`.
- A list can grow and shrink; a string cannot.

The everyday list tools:

Tool	Does
`a.append(x)`	adds `x` at the end
`a.insert(i, x)`	inserts `x` at position `i`
`a.remove(x)`	removes the first `x`
`a.pop()` / `a.pop(i)`	removes and returns the last item / item `i`
`a.sort()`	sorts the list in place
`sorted(a)`	returns a NEW sorted list
`x in a`	is `x` in the list?
`len(a)`, `sum(a)`, `max(a)`, `min(a)`	size and quick maths



Exam tip: first item is index 0; last is len(list)-1; negative indices count from the end

List indices start at 0

Traversing a list

To traverse 遍历 a list is to visit each item. A `for` loop does this with no index needed.

```
scores = [88, 71, 95]
total = 0
for s in scores:
    total = total + s
print(total)          # 254
```

- Use `enumerate` when you also need the index.

```
for i, name in enumerate(["a", "b"]):
    print(i, name)    # 0 a  /  1 b
```

2-D lists (grids)

A 2-D list 二维列表 is a list of lists — a grid 网格 of rows and columns. Use two indexes: `grid[row][col]`.

```
grid = [[1, 2, 3],
        [4, 5, 6]]
print(grid[0][2])     # 3
print(grid[1][0])     # 4
```

- A nested loop 嵌套循环 visits every cell.

```
grid = [[1, 2], [3, 4]]
for row in grid:
    for value in row:
        print(value, end=" ")
print()               # 1 2 3 4
```

List comprehensions

A list comprehension 列表推导式 builds a new list in one line: `[expression for item in sequence]`.

```
squares = [x * x for x in range(5)]
print(squares)        # [0, 1, 4, 9, 16]
```

- Add `if` to keep only some items.

```
evens = [n for n in range(10) if n % 2 == 0]
print(evens)          # [0, 2, 4, 6, 8]
```

Tuples & sets

A tuple 元组 is a fixed sequence in round brackets. It cannot be changed after it is made — use one for values that belong together, and unpack 解包 it into names.

```
point = (3, 4)
x, y = point            # unpacking
print(x, y)             # 3 4
```

A function that needs to hand back two results returns a tuple:

```
def min_max(nums):
    return min(nums), max(nums)

lo, hi = min_max([5, 2, 9])
print(lo, hi)           # 2 9
```

A set 集合 stores each value once, with no order. It is perfect for removing duplicates and for fast membership tests 成员测试.

```
votes = ["red", "blue", "red", "green", "red"]
colours = set(votes)
print(len(colours))         # 3  (duplicates removed)
print("blue" in colours)    # True
```

Common mistakes

- `b = a` does **not** copy a list: both names point to the same list, so changing one changes the other. Use `a.copy()` or `a[:]`.
- The last item is `a[-1]`; `a[len(a)]` is out of range.
- `append` adds ONE item; use `extend` or `+` to join another list.
- Building a grid with `[[0]*3]*3` makes three copies of the **same** row. Build the rows in a loop instead.
- A tuple with one item needs a comma: `(5,)`, not `(5)`.
- A set has no order and no duplicates, so you cannot index it with `s[0]`.