Arrays

Ref Pseudocode

One-dimensional arrays

An **array** 数组 holds many values of one type under one name. Each value sits at an **index** 下标.

```
DECLARE <identifier> : ARRAY[<lower>:<upper>] OF <data type>
```

```
DECLARE Names : ARRAY[1:3] OF STRING
Names[1] ← "Mei"
Names[2] ← "Sam"
Names[3] ← "Ana"
OUTPUT Names[2]
```

- The bounds 边界 are written `lower:upper`, and **both are included** — `[1:3]` has three elements 元素.
- Cambridge arrays normally start at **1**, not 0.
- Every element has the same type, fixed by `OF`.

Filling and reading with a FOR loop

An array and a counted loop belong together: the counter is the index.

```
DECLARE Scores : ARRAY[1:5] OF INTEGER
DECLARE Index, Total : INTEGER
FOR Index ← 1 TO 5
    Scores[Index] ← Index * 10
NEXT Index
Total ← 0
FOR Index ← 1 TO 5
    Total ← Total + Scores[Index]
NEXT Index
OUTPUT "Total: ", Total
OUTPUT "Mean: ", Total / 5
```


Finding the largest

```
DECLARE Scores : ARRAY[1:5] OF INTEGER
DECLARE Index, Largest : INTEGER
Scores[1] ← 42
Scores[2] ← 17
Scores[3] ← 93
Scores[4] ← 8
Scores[5] ← 55
Largest ← Scores[1]
FOR Index ← 2 TO 5
    IF Scores[Index] > Largest
      THEN
        Largest ← Scores[Index]
    ENDIF
NEXT Index
OUTPUT "Largest: ", Largest
```

Start `Largest` at the **first element**, never at 0 — with negative data, 0 would win and the answer would be wrong.

Common mistakes

- Reading `Scores[6]` from an array declared `[1:5]`. That is out of bounds 越界.
- Writing `Scores(3)`. Arrays use square brackets; round brackets call a function.
- Declaring `[1:5]` and then looping `0 TO 4`.

Two-dimensional arrays

A **2-D array** 二维数组 is a table: rows and columns, two indexes.

```
DECLARE <identifier> : ARRAY[<lower1>:<upper1>, <lower2>:<upper2>] OF
↪  <data type>
```

```
DECLARE Grid : ARRAY[1:2, 1:3] OF INTEGER
Grid[1,1] ← 1
Grid[1,2] ← 2
Grid[1,3] ← 3
Grid[2,1] ← 4
Grid[2,2] ← 5
Grid[2,3] ← 6
OUTPUT Grid[2,3]
```

The **first** index is the row 行, the **second** the column 列.

Nested loops walk a table

One loop per dimension 维度. The inner loop finishes a whole row before the outer moves on.

```
DECLARE Grid : ARRAY[1:2, 1:3] OF INTEGER
DECLARE Row, Col : INTEGER
FOR Row ← 1 TO 2
    FOR Col ← 1 TO 3
        Grid[Row, Col] ← Row * Col
    NEXT Col
NEXT Row
FOR Row ← 1 TO 2
    FOR Col ← 1 TO 3
        OUTPUT "Grid[", Row, ",", Col, "] = ", Grid[Row, Col]
    NEXT Col
NEXT Row
```

△ The inner `NEXT` closes the **inner** counter. `NEXT Row` before `NEXT Col` crosses the loops over and is nonsense — a marker spots it instantly.

Common mistakes

- Swapping row and column. `Grid[2,3]` is row 2, column 3.
- `Grid[Row][Col]`. Cambridge writes **one** pair of brackets with a comma inside.
- Crossing the `NEXT` lines over, as above.