Exercise walk-through

Arrays ▪ 8.2

eXercise

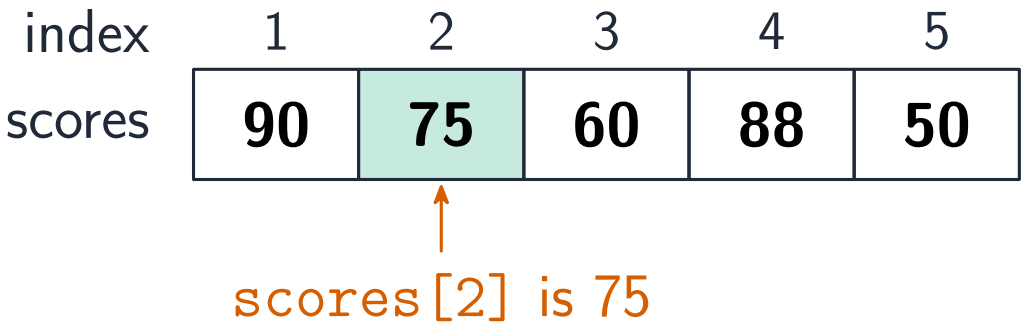
You must be able to

- declare a 1D and a 2D array and use an **index 索引** to read/write a value
- use a loop (and a nested loop) to process an array
- choose between a 1D and a 2D array for a problem

Method — 1D and 2D arrays

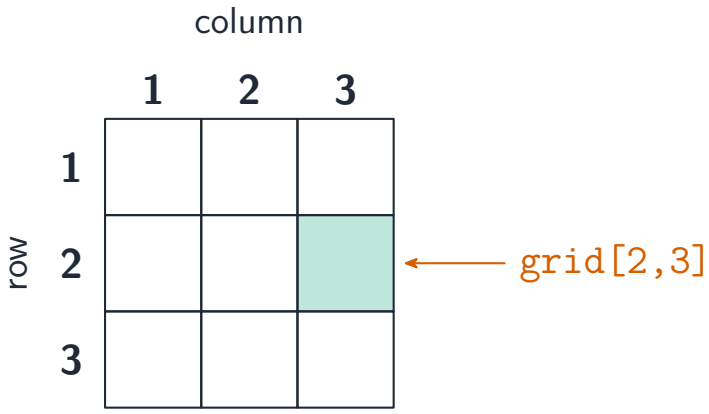
- 1D: `DECLARE scores : ARRAY[1:5] OF INTEGER`, then `scores[2] ← 75`.
- 2D: `DECLARE grid : ARRAY[1:3, 1:3] OF INTEGER`, then `grid[2, 3] ← 8`.

Method — 1D and 2D arrays



Five indexed boxes; index 2 holds 75

Method — 1D and 2D arrays



A 3 by 3 grid; the cell at row 2, column 3 is highlighted

Method — Arrays, files, and the tests that find the bugs

A one-dimensional array 一维数组 is a list of values under one name, each reached by an index: `DECLARE Scores : ARRAY[1:10] OF INTEGER`, so `Scores[3]` is the third value. **A two-dimensional array 二维数组** is a table with a row and a column index: `DECLARE Grid : ARRAY[1:5, 1:4] OF STRING`, so `Grid[2,3]` is row 2, column 3. Use a nested loop to work through a 2D array: the outer loop over rows, the inner over columns.

Reading and writing a file in Cambridge pseudocode:

```
OPENFILE "Marks.txt" FOR WRITE
WRITEFILE "Marks.txt", Name
CLOSEFILE "Marks.txt"

OPENFILE "Marks.txt" FOR READ
READFILE "Marks.txt", Line
CLOSEFILE "Marks.txt"
```

Method — Arrays, files, and the tests that find the bugs

Always close a file: leaving it open can lose the data that has not yet been written.
Validation 验证 checks that the data is **sensible** as it is entered, and the computer can do it: a **range check** (is the mark between 0 and 100?), a **length check** (is the password at least 8 characters?), a **type check** (is it a whole number?), a **presence check** (was anything entered?), a **format check** (does the date look like DD/MM/YYYY?), and a **check digit** (does the extra digit match a calculation on the others?).
Verification 校验 checks that the data has been **copied correctly**: **double entry** (type it twice and compare) or a **visual check** (read it against the original).

Method — Arrays, files, and the tests that find the bugs

Test data. For a mark out of 100: **normal** data such as 45 is accepted; **abnormal** data such as −5 or `A` is rejected; **extreme** (boundary) data such as 0 and 100 is accepted, while 101 is rejected. Write a test plan as a table of the value, the reason for choosing it and the expected result.

Worked example. State one validation and one verification check for a password, with a reason. Validation: a length check that the password is at least eight characters, because a short password is easily guessed. Verification: double entry, typing it twice and comparing, because a typing mistake would lock the user out of their own account.

Practice 2.1 ▪ 1 mark

An array is declared `DECLARE temps : ARRAY[1:7] OF REAL`. State how many values it can hold.

Solution 2.1

Method: Arrays, files, and the tests that find the bugs

Worked steps

7 B1.

Practice 2.2 ▪ 2 marks

State one difference between a 1D and a 2D array.

Solution 2.2

Method: Arrays, files, and the tests that find the bugs

Worked steps

a 1D array is a single list using one index M1; a 2D array is a table using two indexes [row, column] A1.

Practice 2.3 ▪ 2 marks

Give the pseudocode declaration for an array holding 20 real numbers.

Solution 2.3

Method: Arrays, files, and the tests that find the bugs

Worked steps

`DECLARE Numbers : ARRAY[1:20] OF REAL` B1 B1 *for the array with bounds, for the type.*

Practice 2.4 ▪ 3 marks

Identify the type of validation check for each: a date entered as DD/MM/YYYY; an age between 0 and 120; a name box that must not be left empty.

Solution 2.4

Method: Arrays, files, and the tests that find the bugs

Worked steps

Format check B1; range check B1; presence check B1.

Practice 2.5 ▪ 2 marks

Explain the difference between validation and verification.

Solution 2.5

Method: Arrays, files, and the tests that find the bugs

Worked steps

Validation checks that the data entered is sensible or reasonable, and the computer performs it automatically B1; verification checks that the data has been copied or entered accurately compared with the original B1.

Exam-style 3.1 ▪ 1 mark

Which data structure best stores the marks of 30 students in **one** subject?

A a single variable

B a 1D array

C a 2D array

D a constant

Solution 3.1

Method: Arrays, files, and the tests that find the bugs

A a single variable

B a 1D array ✓

C a 2D array

D a constant

Worked steps

B. One subject = a single list = a 1D array.

Exam-style 3.2 ▪ 3 marks

An array `names` holds 5 names in elements 1 to 5.
(a) Write pseudocode to output every name using a loop.
(b) State which type of array (1D or 2D) you would use to store the marks of 30 students in **5** subjects, and why.

Solution 3.2

Method: Arrays, files, and the tests that find the bugs

Worked steps

(a) `FOR i ← 1 TO 5` M1; `OUTPUT names[i]` M1; `NEXT i` A1.
(b) a 2D array M1; because each student needs 5 marks, so two indexes [student, subject] are required A1.

Exam-style 3.3 ▪ 2 marks

A program stores the marks of 30 students in a one-dimensional array called `Marks`, out of a maximum of 100.
(a) Write the pseudocode declaration for the array.
(b) Write pseudocode that inputs 30 marks into the array, rejecting any mark that is not between 0 and 100 and asking again.
(c) Complete the test plan by giving one value of each type and the expected result.

Type of test data	**Value**	**Expected result**
normal		
abnormal		
extreme		

Exam-style 3.3 ▪ 2 marks

(d) The marks are then written to a file. Write the three lines of pseudocode that open the file for writing, write `Marks[1]` and close it.
(e) Explain why the file must be closed.
(f) The teacher's name is typed twice when the program starts. Identify this check and explain its purpose.

Solution 3.3

> Method: Arrays, files, and the tests that find the bugs

Worked steps

(a) `DECLARE Marks : ARRAY[1:30] OF INTEGER` B1 B1 *bounds, type.*
(b) `FOR Count ← 1 TO 30` with `NEXT Count` B1; a loop that repeats the input while the value is out of range, such as `REPEAT … UNTIL Mark >= 0 AND Mark <= 100` B1 B1 *for the correct condition*; `INPUT Mark` inside that loop B1; an error message when the value is rejected B1; `Marks[Count] ← Mark` inside the outer loop B1.
(c) Normal: 45, accepted B1. Abnormal: -5 or `A`, rejected with an error message B1. Extreme: 0 or 100, accepted; 101, rejected B1.
(d) `OPENFILE "Marks.txt" FOR WRITE` B1; `WRITEFILE "Marks.txt",`

Solution 3.3

`Marks[1]` B1; `CLOSEFILE "Marks.txt"` B1.
(e) Data still held in memory may not have been written to the disc, so closing the file makes sure it is saved B1.
(f) Double entry, a verification check B1; it catches a typing mistake, because a slip is unlikely to be repeated identically the second time B1.