AQA AS Computer Science – Paper 1 (7516/1)

Worked Solutions & Mark Scheme · Tuesday 12 May 2026

Total: 75 marks · Time: 1 h 45 min. Mark codes: **B** = independent mark, **M** = method mark, **A** = accuracy mark.

Important – what is and isn't fully solved here. Paper 1 is partly based on the **Preliminary Material and Skeleton Program**, which were not in the files you uploaded and which are not publicly available for the 2026 series. So:
• **Section A (Q01–Q03)** is self-contained and is **fully solved and verified** (code was executed).
• **Section B (Q04–Q10)**: answers that do not depend on the source are given in full. Parts that need the actual code are marked **[NEEDS SKELETON]** – I give a provisional answer or a method for finding it; please verify these against your copy.
• **Section C (Q11–Q13)**: reference solutions are written in **Python** using **assumed** identifiers (Board, Width, Height, TILE, NO_TILE, ProcessMove…). The logic and mark scheme are right, but names/indexing must be adapted to the real skeleton. If you paste the skeleton code (or upload it) I can make these exact.

This is an unofficial, self-written mark scheme in AQA style; not AQA's published scheme.

Section	Questions	Marks	Status
A	01–03	≈ 20	Fully solved
B	04–10	≈ 25	Partly needs Skeleton Program
C	11–13	≈ 30	Reference code with assumed names

Section A

Q01.1 Hand-trace IssueChange(24, 100); complete Table 1 (change, i, coinValues[i]). [5 marks]

ANSWER / WORKED SOLUTION

change = paid − cost = 100 − 24 = **76**. coinValues = [50, 20, 10, 5, 2, 1]. For each i the check *change* ≥ *coinValues[i]* is made and, if true, the coin is subtracted.

change	i	coinValues[i]	Comment
76			change initialised
76	0	50	76 ≥ 50 → change becomes 26
26	1	20	26 ≥ 20 → change becomes 6
6	2	10	6 < 10 → no coin
6	3	5	6 ≥ 5 → change becomes 1
1	4	2	1 < 2 → no coin
1	5	1	1 ≥ 1 → change becomes 0
0			final value returned

Coins released: 50p, 20p, 5p, 1p (= 76p). The returned value 0 means all change was issued.

MARK SCHEME

Mark	Award the mark for…
B1	First row: change = 76 (i and coinValues[i] blank).
B1	i = 0 / coinValues = 50 and i = 1 / coinValues = 20 with change 76 then 26.
B1	i = 2 / 10 with change 6 and i = 3 / 5 with change 6.
B1	i = 4 / 2 with change 1 and i = 5 / 1 with change 1.

Mark	Award the mark for…
B1	Final change = 0.
	Mark values in the correct columns/rows in order. Accept the final 0 on its own last row. Ignore any extra comment column.

Q01.2 Give one call to IssueChange (both parameters positive, < 100) that leaves some change unissued and explain why. **[3 marks]**

ANSWER / WORKED SOLUTION

Example: IssueChange(1, 5) → change = 4.
Trace: 4 < 50, 20, 10, 5; 4 ≥ 2 → release 2p, change = 2; 2 ≥ 1 → release 1p, change = 1. The algorithm returns **1**, not 0.
Why: the algorithm uses a **FOR loop with an IF**, so each coin value is considered **only once** and at most one coin of each denomination can be issued. Change of 4p needs *two* 2p coins, but only one 2p coin is released, leaving 1p unissued. (Any change needing the same coin twice works, e.g. 40p needs 20p+20p.)

MARK SCHEME

Mark	Award the mark for…
B1	A valid call where the change amount needs two or more coins of the same value (e.g. (1,5), (10,50) → 40, (5,15) → 10). Both parameters < 100, positive, paid ≥ cost.
B1	Each coin value is only checked/issued once (IF not WHILE / one pass through the list).
B1	Explains that the amount requires more than one coin of that value, so the remainder is not issued.
	Check any example by tracing: it must return a non-zero value. e.g. change 1–98 that is not made of distinct denominations.

Q02 State two examples of tasks undertaken during the analysis stage of software development. **[2 marks]**

ANSWER / WORKED SOLUTION

Any two: • Interview/survey/observe users and stakeholders to find out what they need. • Identify the problem and define its scope/boundaries. • Produce a requirements specification. • Carry out a feasibility study (cost, time, technical). • Identify constraints and existing systems/data. • Model data flow/processes (e.g. data flow diagrams).

MARK SCHEME

Mark	Award the mark for…
B1	Gathering requirements (interviews, questionnaires, observation).
B1	Producing a requirements specification / defining the problem or scope / feasibility study / modelling the existing system.
	Max 2. Design, coding and testing tasks are not creditworthy.

Q03.1 Write a program to implement the algorithm in Figure 2 (lowest common multiple). **[9 marks]**

ANSWER / WORKED SOLUTION

The algorithm finds the **lowest common multiple** m of in1 and in2 by counting up from 1 until m is divisible by both, then outputs m ÷ in1, m ÷ in2 and m. Python solution (tested): note // is used so that whole numbers (11, 8, 176) are displayed.

```
# Task 1 - Python (MOD is %)
in1 = int(input("Enter the first number: "))
in2 = int(input("Enter the second number: "))

if in1 < 1 or in2 < 1:
    print("Invalid input")
else:
    m = 1
    found = False
    while found == False:
        if m % in1 == 0 and m % in2 == 0:
            found = True
        else:
            m = m + 1
    print(m // in1)
    print(m // in2)
    print(m)
```

MARK SCHEME

Mark	Award the mark for…
B1	Both prompts output with the exact text and both values input (converted to integers where required).
B1	Selection with condition testing in1 < 1 OR in2 < 1 (both tested, logical OR).
B1	"Invalid input" output in the true branch (and nothing else is run).
B1	Else branch: m initialised to 1 and found to False.
B1	Indefinite iteration (WHILE) controlled by found = False / not found.
B1	Condition uses the MOD/% operator correctly for both numbers with AND.
B1	found set to True when condition met; else m incremented by 1.
B1	Outputs m ÷ in1 and m ÷ in2 (whole-number result or accepted real, e.g. 11.0).
B1	Outputs m after the loop; whole program structure/indentation correct and it runs.
	Max 9. Penalise: INPUT stored as strings with no conversion (program fails), missing ELSE (outputs after "Invalid input"), outputs inside the loop.

Q03.2 Screen capture of the test (16 and 22). **[1 mark]**

ANSWER / WORKED SOLUTION

Run the program, type 16 then 22. Expected screen:

```
Enter the first number: 16
Enter the second number: 22
11
8
176
```

Check: LCM(16, 22) = 176; 176 ÷ 16 = 11; 176 ÷ 22 = 8.

MARK SCHEME

Mark	Award the mark for…
B1	Screen capture showing inputs 16 and 22 and the outputs 11, 8 and 176 (11.0 and 8.0 accepted).

Section B – Preliminary Material / Skeleton Program

Questions in this section refer to the Skeleton Program. Items tagged **[NEEDS SKELETON]** cannot be answered with certainty from the question paper alone.

Q04.1 Identifier of a variable used to store a Boolean value. **[1 mark]**

[NEEDS SKELETON] How to find it: search the skeleton for variables assigned True/False (or declared Boolean), typically a flag controlling a loop or recording whether a move was valid. Write the identifier exactly as spelt/cased in your language.

MARK SCHEME

Mark	Award the mark for…
B1	Identifier of a variable that is assigned/declared as Boolean in the skeleton (spelling and case as in the code).

Q04.2 Identifier of a user-defined subroutine that returns a string. [1 mark]

[NEEDS SKELETON] Look for a **function** declared with a string return type/return statement that returns a string. Since ConvertCoordsToRef builds a reference (e.g. “B3”) from coordinates, it is a strong candidate – confirm in the code.

MARK SCHEME

Mark	Award the mark for…
B1	Name of a user-defined function whose return value is a string.

Q04.3 Identifier of a user-defined subroutine containing nested indefinite iteration. [1 mark]

[NEEDS SKELETON] Look for a WHILE/REPEAT/DO loop that has **another** WHILE/REPEAT/DO loop inside it within the same subroutine.

MARK SCHEME

Mark	Award the mark for…
B1	Name of a subroutine containing an indefinite loop nested within another indefinite loop.

Q05 Explain the difference between definite and indefinite iteration. [2 marks]

ANSWER / WORKED SOLUTION

Definite iteration: the number of repetitions is **known/fixed before the loop starts** (e.g. a FOR loop running 10 times). **Indefinite iteration**: the number of repetitions is **not known in advance** – the loop repeats while/until a **condition** is met (e.g. WHILE or REPEAT-UNTIL).

MARK SCHEME

Mark	Award the mark for…
B1	Definite – loop executes a set/known number of times (counted loop, FOR).
B1	Indefinite – repeats until/while a condition is true; number of iterations not known beforehand (WHILE/REPEAT).

Q06 State one advantage and two disadvantages of using global variables. [3 marks]

ANSWER / WORKED SOLUTION

Advantage (one): accessible from every subroutine, so data need not be passed as parameters – less code and easy sharing of data; they keep their value for the whole run.
Disadvantages (two): any subroutine can change them, causing **unintended side effects** that are hard to trace/debug; subroutines become **dependent on the globals**, reducing reusability/modularity and making testing harder; they occupy memory for the entire program run; risk of name clashes.

MARK SCHEME

Mark	Award the mark for…
B1	Advantage: accessible anywhere / no need to pass as parameters / simpler to share data.
B1	Disadvantage 1: may be changed accidentally/unexpectedly by any subroutine – side effects, harder to debug.
B1	Disadvantage 2: reduces reusability/modularity/testability of subroutines, or uses memory for the whole run.

Mark	Award the mark for…
	Max 1 advantage + max 2 disadvantages.

Q07.1 Probability of NO_TILE being returned by GetRandomTile. **[1 mark]**

[NEEDS SKELETON] Method: P(NO_TILE) = (number of random values that lead to NO_TILE) ÷ (total number of values the generator can return). Read the range and the comparison in GetRandomTile. *Illustration only:* if the generator returned 0–4 and TILE were returned for values 0–2, then P(NO_TILE) = 2/5.

MARK SCHEME

Mark	Award the mark for…
B1	Correct fraction/decimal/percentage computed from the actual range and condition in GetRandomTile.

Q07.2 Describe a change to GetRandomTile that increases the probability of NO_TILE. **[2 marks]**

ANSWER / WORKED SOLUTION

Change the random-number test so that a **larger share of the possible values** produce NO_TILE: e.g. **increase the upper limit of the random range** (without changing the TILE condition) or **change the comparison value/threshold** so fewer numbers return TILE. Example: if TILE is returned when the number is below a threshold, lower that threshold.

MARK SCHEME

Mark	Award the mark for…
B1	Modify the range of the random number generator or the value used in the selection/condition.
B1	In a way that makes more of the possible values give NO_TILE (explained, not just “change the code”).

Q08.1 What does each rectangle in a hierarchy chart represent? **[1 mark]**

ANSWER / WORKED SOLUTION

A **subroutine** (procedure/function/module) in the program.

MARK SCHEME

Mark	Award the mark for…
B1	A subroutine / procedure / function / module.

Q08.2 / 08.3 Write the subroutines for rectangles (a) and (b) in Figure 3. **[2 marks]**

[NEEDS SKELETON] Main's children appear in the chart as (a), LoadTestBoard, ResetBoard, SetBoardSize, (b). From the paper's own menu options (1 = start game, 2 = set board size, 4 = load test board) and the rest of the paper, the likely answers are: **(a) = the subroutine that displays the menu** (e.g. a DisplayMenu-type routine) and **(b) = PlayGame** (whose four children would be the routines it calls such as the board display, move input and ProcessMove). Confirm exact identifiers in the skeleton: (b) is the subroutine with four child subroutines under it.

MARK SCHEME

Mark	Award the mark for…
B1	(a) the correct subroutine called directly from Main (menu display).
B1	(b) the correct subroutine called directly from Main with four called subroutines (PlayGame).

Q09.1 How would ProcessMove handle three references such as C1-C2-C3? **[1 mark]**

[NEEDS SKELETON] Check how the input is split on the dash. Typical outcome: only the first and second references (or the first two parts) are used, so the third reference is **ignored** / the move is treated as C1-C2 (or rejected as invalid if the code checks the number of parts). State precisely what the code does.

MARK SCHEME

Mark	Award the mark for…
B1	Correct description of the subroutine's behaviour: the third reference is ignored/not used (or the move is rejected), consistent with the split/length logic in the code.

Q09.2 Why does A1-A10 behave identically to A1 on a 10 × 10 board? **[2 marks]**

[NEEDS SKELETON] Likely explanation (verify): references are converted using only the **first character (column letter) and the second character (row digit)**, so “A10” is read as row “1”, i.e. A1. Then A1-A10 becomes A1-A1, a range containing only the single square A1, which is exactly the same as entering A1.

MARK SCHEME

Mark	Award the mark for…
B1	The second reference A10 is read as A1 (only one digit/character used for the row).
B1	So the range is A1-A1 – start and end are the same single square, as for input A1.

Q09.3 Purpose of the third selection statement in ProcessMove. **[2 marks]**

[NEEDS SKELETON] Identify the third IF/selection in ProcessMove in order of appearance, then say what condition it tests and what happens (typically validating the move: e.g. checking the start/end references are on the board or in the same row/column, and what is done if not). Award: 1 mark for what is tested, 1 mark for the consequence/purpose.

MARK SCHEME

Mark	Award the mark for…
B1	Correctly identifies what the condition checks.
B1	Correctly states the effect/purpose (e.g. marks the move invalid / prevents out-of-range or diagonal moves / only removes tiles when legal).

Q09.4 Describe how the variable ToRemove is used in ProcessMove. **[4 marks]**

[NEEDS SKELETON] Trace ToRemove from declaration to last use. A typical 4-mark answer covers: (1) it is declared/initialised (e.g. empty or 0); (2) it is added to/updated during a loop over the squares in the move range, recording the squares/tiles to be removed; (3) it is checked/used in a condition to decide whether the move is valid (e.g. is not empty or equals the move length); (4) it is used to clear/remove those tiles and/or to report how many were removed.

MARK SCHEME

Mark	Award the mark for…
B1	Initialised before use.
B1	Updated/appended inside the loop that processes the squares in the move.
B1	Used in a condition to determine validity.
B1	Used to remove tiles / return or display the result.
	Mark points are indicative until checked against the code.

Q10 Describe how ConvertRefToCoords converts the reference C6 to column 2 and row 5. **[2 marks]**

ANSWER / WORKED SOLUTION

Column: take the first character, **C**, convert it to its character code (ASCII/Unicode, 67) and subtract the code of **A** (65) → 67 − 65 = **2**.

Row: take the second character, **6**, convert it to an integer (6) and subtract **1** → **5** (because rows/columns are counted from 0).

MARK SCHEME

Mark	Award the mark for…
B1	Column: character code of the letter minus the code of A (C → 2).
B1	Row: the digit converted to an integer and 1 subtracted (6 → 5).

Section C – Programming (reference solutions)

Assumed skeleton details (adapt to your copy): global Board as a list of rows, Board[row][col]; globals Width and Height; constants TILE and NO_TILE; ResetBoard() rebuilds the board from Width/Height; ConvertCoordsToRef(col, row) returns a reference such as "B3"; ProcessMove(move) returns True for a valid move and removes the tiles. If your skeleton differs (e.g. Board[col][row], or ProcessMove returns nothing), change those lines only – the algorithm and mark scheme stay the same.

Q11.1 Amend SetBoardSize: width and height must each be 3–9; otherwise leave board unchanged and show the error message. **[6 marks]**

```
def SetBoardSize():
    global Width, Height
    newWidth = int(input("Enter width: "))
    newHeight = int(input("Enter height: "))
    if newWidth >= 3 and newWidth <= 9 and newHeight >= 3 and newHeight <= 9:
        Width = newWidth
        Height = newHeight
        ResetBoard()
    else:
        print("Invalid input – size of board unchanged")
```

Key idea: read both values into **temporary** variables first and only copy them into the real board size when *both* pass validation – that guarantees the size is unchanged on invalid input.

MARK SCHEME

Mark	Award the mark for…
B1	Width and height both input (stored temporarily / original size not overwritten before validation).
B1	Lower bound test: each value ≥ 3.
B1	Upper bound test: each value ≤ 9.
B1	Both width AND height must pass (AND combining the four tests / correct logic).
B1	Valid: board size set using the new values (and board reset/re-created as the skeleton requires).
B1	Invalid: exact message "Invalid input – size of board unchanged" displayed and size not changed.
	The "2, 9, 10 → invalid; 2, 3, 3 → valid" test must work.

Q11.2 Screen capture of the test. **[1 mark]**

ANSWER / WORKED SOLUTION

Select option 2: width 9, height 10 → Invalid input – size of board unchanged (board size stays as before).
Select option 2 again: width 3, height 3 → the board is now 3 × 3.

MARK SCHEME

Mark	Award the mark for…
B1	Screen capture(s) showing the invalid-input message after 9 and 10, and a 3 × 3 board after entering 3 and 3.

Q12.1 Create the subroutine InvertBoard. **[5 marks]**

```
def InvertBoard():
    for row in range(Height):
        for col in range(Width):
            if Board[row][col] == TILE:
                Board[row][col] = NO_TILE
            else:
                Board[row][col] = TILE
```

MARK SCHEME

Mark	Award the mark for…
B1	Subroutine named InvertBoard defined (no parameters needed).
B1	Iterates over every row (using the board height) …
B1	… and, nested within it, every column (using the board width).
B1	Selection tests whether the square contains a tile.
B1	Tile → empty and empty → tile (both cases handled, each square changed once).

Q12.2 Amend PlayGame so that entering “invert” calls InvertBoard and play passes to the next player. **[2 marks]**

```
# inside PlayGame, after the move text has been read:
move = input("Enter move: ")
if move == "invert":
    InvertBoard()
else:
    ProcessMove(move)      # existing handling (validation etc.)
# ...then the existing code that switches to the other player
```

MARK SCHEME

Mark	Award the mark for…
B1	Test for the text "invert" (before the move is processed/validated as a reference).
B1	InvertBoard called and the turn then passes to the next player (existing player-switch code is still executed; ProcessMove not run for “invert”).

Q12.3 Screen capture of the test. **[1 mark]**

ANSWER / WORKED SOLUTION

Option 4 loads the test board, then invert. Every X becomes empty and every empty square becomes X (as in Figure 4 → Figure 5), and the next player's turn is shown.

MARK SCHEME

Mark	Award the mark for…
B1	Screen capture showing the board before and after (or after) inversion with all squares swapped.

Q13.1 Create PlayRandomMove. **[12 marks]**

```
import random

def PlayRandomMove():
    validMove = False
    while not validMove:
        startCol = random.randint(0, Width - 1)          # random start square
        startRow = random.randint(0, Height - 1)
        if random.randint(0, 1) == 0:                     # horizontal (1 in 2)
            endCol = random.randint(startCol, Width - 1)  # at or to the right
            endRow = startRow
        else:                                             # vertical
            endRow = random.randint(startRow, Height - 1) # at or below
            endCol = startCol
        startRef = ConvertCoordsToRef(startCol, startRow)
        endRef = ConvertCoordsToRef(endCol, endRow)
        move = startRef + "-" + endRef

        # note which squares hold tiles BEFORE the move (to display correctly)
        tiles = []
        for r in range(startRow, endRow + 1):
            for c in range(startCol, endCol + 1):
                if Board[r][c] == TILE:
                    tiles.append((c, r))

        validMove = ProcessMove(move)                     # repeat until valid

    if len(tiles) > 1:
        print(move)                                       # range, e.g. B3-B4
    else:
        c, r = tiles[0]
        print(ConvertCoordsToRef(c, r))                   # single reference, e.g. A5
```

MARK SCHEME

Mark	Award the mark for…
B1	Subroutine PlayRandomMove with no parameters and no return value.
B1	Random start column and row chosen within the board limits (using width/height).
B1	Random choice of horizontal/vertical with equal (1 in 2) probability.
B1	Horizontal: end row = start row.
B1	Horizontal: end column random from start column to last column (at/right of start).
B1	Vertical: end column = start column.
B1	Vertical: end row random from start row to last row (at/below start).
B1	ConvertCoordsToRef called for both start and end squares (coordinates → references).
B1	Move built by concatenating start reference, a dash and end reference.
B1	Move passed to ProcessMove.
B1	Whole selection/building/ProcessMove process repeated (indefinite loop) until a valid move is made.
B1	Displays the move: range when more than one tile removed; single reference when one tile removed.
	Max 12. Do not penalise a different but equivalent approach to counting tiles (e.g. comparing tile counts before/after the move).

Q13.2 Amend PlayGame to call PlayRandomMove when “random” is entered. **[1 mark]**

```
if move == "random":
    PlayRandomMove()
elif move == "invert":       # from Q12 (if kept)
    InvertBoard()
else:
    ProcessMove(move)
```

MARK SCHEME

Mark	Award the mark for…
B1	Selection on input text "random" calling PlayRandomMove (and normal moves still work).

Q13.3 / 13.4 Screen captures of the tests. **[2 marks]**

ANSWER / WORKED SOLUTION

Task 3: start a new game (1), then enter random: a randomly generated legal move is displayed (e.g. B3-B4 or a single reference) and the board updates. Results vary between runs because the move is random.

Task 4: load the test board (4), then enter random: a legal random move is displayed and the tiles it covers are removed from the test board.

MARK SCHEME

Mark	Award the mark for…
B1	13.3 – screen capture of a game started with option 1 and “random” entered, showing a displayed move and updated board.
B1	13.4 – screen capture of the test board (option 4) with “random” entered, showing a displayed move and updated board.