

Exam technique

How the exams work, technique for every type of question, and every command word

How the exams work

OCR GCSE Computer Science (J277) is assessed by two written exams at the end of the course, each 1 hour 30 minutes and 80 marks and each worth half of the GCSE. There is one tier (grades 9 to 1) and no calculator is allowed in either paper. Code in the questions is usually written in OCR Exam Reference Language (ERL), and the programming project you do in lessons is not marked, so all your programming skills are tested on paper in Paper 2.

Paper	Time	Marks	Calculator	What's on it
Paper 1 (J277/01): Computer systems	1 h 30 min	80	Not allowed	1.1 Systems architecture, 1.2 Memory and storage, 1.3 Computer networks, connections and protocols, 1.4 Network security, 1.5 Systems software, 1.6 Ethical, legal, cultural and environmental impacts of digital technology. Short and medium-length questions (including calculations) across all six topics, usually including an extended response question marked by levels.
Paper 2 (J277/02): Computational thinking, algorithms and programming	1 h 30 min	80	Not allowed	2.1 Algorithms, 2.2 Programming fundamentals, 2.3 Producing robust programs, 2.4 Boolean logic, 2.5 Programming languages and Integrated Development Environments. Section A covers these topics. Section B has longer questions where you write, test and refine algorithms and programs; in Section B, algorithms must be written in OCR Exam Reference Language or a high-level programming language you have studied.

Exam technique

Preparing in the weeks before

- Revise the two papers differently. Paper 1 (1.1 to 1.6) is mostly knowledge, explanation and calculation. Paper 2 (2.1 to 2.5 and Section B) is mostly reading, tracing and writing algorithms and code.
- Make flashcards for the definitions that come up again and again: volatile and non-volatile, cache, virtual memory, abstraction, decomposition, protocol, topology, compiler, interpreter, syntax error, logic error, and the job of each register (PC, MAR, MDR, accumulator).
- Practise binary, hexadecimal and file-size calculations by hand every few days. There is no calculator in either paper, so working with 8-bit numbers has to be quick and accurate.
- Write code by hand, not only on a computer. In the exam every program is written on paper with no IDE to catch your mistakes, so practise indenting clearly, closing every `if`, loop and sub program, and checking your own code with a trace table.
- Learn to read OCR Exam Reference Language (ERL) fluently, because the code in the questions is usually written in it: `for i = 1 to 5 ... next i` (which includes 5), `while ... endwhile`, `do ... until`, `if ... then ... elseif ... then ... else ... endif`, `switch ... case ... endswitch`, `.length`, `.substring(start, number of characters)`, `MOD`, `DIV` and `^`.
- Learn the exact names of the three laws in 1.6 (the Data Protection Act 2018, the Computer Misuse Act 1990 and the Copyright, Designs and Patents Act 1988) and two or three specific things each one requires or makes illegal.
- Write at least three timed extended response answers on different scenarios before the exam, and mark them against a levels mark scheme.
- In the last few weeks, sit full papers in 90 minutes, one Paper 1 and one Paper 2 at a time, so you know how the time feels.

The night before and the morning of the exam

- Check which paper it is. Paper 1 and Paper 2 test completely different content, so make sure your final review is for the right one.
- The night before, skim your cheatsheets and flashcards: registers, protocols and their jobs, the file-size formulas, the types of test data, the threats and their prevention, and the three laws. Don't start new topics.
- Pack at least two black pens, a pencil, a ruler and an eraser. Write your answers in black ink; use pencil only for diagrams such as logic diagrams and network drawings. Calculators are not allowed.
- Get a proper night's sleep. Tracing code and doing binary arithmetic in your head are the first things to go wrong when you are tired.
- In the morning, eat breakfast and warm up for five minutes: convert a number between denary, binary and hex and trace a short loop, so you start the paper already thinking clearly.

The first five minutes

- Fill in the front cover, then look through the whole paper. Find the long questions: the extended response question on Paper 1, and Section B on Paper 2.
- On Paper 2, look through Section B quickly and note how many marks it carries. Your brain will keep thinking about it while you work through Section A.
- If it helps, jot down on the question paper (not in an answer space) the things you might blank on: the place values 128 64 32 16 8 4 2 1, the hex digits A = 10 to F = 15, and the file-size formulas.
- Start at question 1 and work in order, but don't get stuck: if a question stops you for more than a minute, mark it with a star and move on.

Timing: about a minute per mark

- Each paper is 80 marks in 90 minutes. Allow about one minute per mark; that leaves roughly 10 minutes to check your work and go back to anything you skipped.
- Let the marks set the length: a 1-mark 'state' question needs under a minute; a 4-mark 'explain' question needs about 4 minutes and two developed points.
- Paper 1: make sure you reach the extended response question with at least a minute per mark left for it, plus 2 minutes to plan.
- Paper 2: add up the marks in Section B and aim to start it with a little more than a minute per mark left, because writing code by hand takes longer than you expect.
- At the halfway point (45 minutes) you should be about halfway through the marks. If you are well behind, speed up on short questions rather than losing a whole long one.
- If a question has taken twice its marks in minutes, write down what you have, mark it and move on.

Reading the question

- Find the command word first. 'State' needs a fact; 'describe' needs what happens; 'explain' needs a reason; 'compare' needs both things; 'discuss' and 'evaluate' need both sides and a judgement.
- Check how many things are asked for, such as 'two reasons' or 'one advantage', and give exactly that many. Extra answers do not earn extra marks, and one that contradicts a correct answer can cost you the mark.
- Look for the context: 'for this school', 'on this network', 'in this game'. When there is a scenario, full marks usually need your answer to use it.
- Look for restrictions: 'other than a firewall', 'using the table', 'using iteration', 'you must use the variable total'. Ignoring them usually loses the mark.
- Note the unit asked for (bits, bytes, kilobytes, megabytes) and any language requirement, such as 'using OCR Exam Reference Language or a high-level programming language'.
- When there is code in the question, read it line by line using the line numbers, and work out what each variable holds before you answer.

Short-answer questions: state, identify, give and tick

- Give one precise fact per mark. 'ROM is non-volatile' earns the mark; 'ROM keeps stuff' may not.
- Use the specification's technical terms: fetch-execute cycle, memory address register, lossless compression, brute-force attack, wireless access point, client-server.
- Words like faster, cheaper, easier or better are not enough on their own. Say what is faster and compared with what: 'A solid state drive has faster read and write speeds than a magnetic hard disk drive.'
- Tick-box questions: tick exactly as many boxes as the question asks. An extra tick usually scores nothing for that row. To change an answer, cross out the wrong tick clearly and tick the right box.
- Table and gap-fill questions: use the words from the list if one is given, and write technical terms clearly so they can't be misread.
- Don't leave any blank. A short, sensible answer using the right technical term can pick up a mark; a blank never does.

Describe, explain and compare questions

- Describe means what something is or what happens, step by step. Explain means why or how, so every point needs a reason.
- For explain, give a point and then expand it with 'because', 'so' or 'which means'. On a 2-mark explain, one mark is usually for the point and one for the reason.
- Use the marks to plan: a 4-mark explain usually needs two points that are each explained, or four linked steps.
- Weak: 'Cache makes the computer faster.' Strong: 'Cache stores frequently used instructions, so the CPU can fetch them from cache instead of slower RAM, which means instructions are processed more quickly.'
- For compare, mention both things in each point and use a comparison word such as 'whereas', 'more' or 'less'. Describing one thing and then the other separately often loses marks.
- Describe a process (the fetch-execute cycle, how DNS finds a web server's IP address, how a binary search works) in the right order, one step per mark, naming each component or register as you go.
- Apply your answer to the scenario. If the question is about a hospital's network, say why it matters to the hospital (patient records, staff logins), not just what the technology does in general.

The extended response question on Paper 1

- Paper 1 usually includes an extended response question (often worth 8 marks) that is marked by levels, not by counting points. The examiner reads the whole answer and judges its overall quality.
- It is usually a discussion of a scenario, often about ethical, legal, cultural, environmental and privacy issues (1.6), and it may bring in technical content from other topics such as security, networks or storage.
- The top level needs accurate technical knowledge, applied to the scenario throughout, and a well-developed line of reasoning that looks at more than one side.
- Spend 1 to 2 minutes planning. List the angles you will cover (for example legal, ethical, environmental, privacy; or benefits and drawbacks for each group of people) and three or four points to develop.
- If the question lists points to consider, cover every one of them. Leaving one out makes the top level very hard to reach.
- Develop every point: make the point, explain it, then give the consequence for someone in the scenario. Three or four well-developed points are worth far more than eight one-line points.
- Name laws exactly and say how they apply, for example: 'Under the Data Protection Act 2018 the company must keep customers' personal data secure and use it only for the purpose it was collected for.'
- Think about different stakeholders: users, employees, the organisation, wider society and the environment.
- If the command word is discuss or evaluate, finish with a short conclusion that weighs up your points and gives a judgement linked to the scenario.
- Write in paragraphs, not bullet points, so the examiner can follow your line of reasoning.

Calculations without a calculator

- The numbers in both papers are chosen so you can work without a calculator. Show every step: working can earn a mark even if the final answer slips.
- Denary to binary: write the place values 128 64 32 16 8 4 2 1 and work from the left. If the question asks for 8-bit binary, give all 8 bits, including leading zeros.
- Binary to hexadecimal: split the byte into two nibbles and convert each one, e.g. 1011 0110 = B6. Denary to hex: go through binary, or divide by 16, e.g. $200 = 12 \times 16 + 8 = \text{C8}$.
- Binary addition: work from right to left; $1 + 1 = 0$ carry 1, and $1 + 1 + 1 = 1$ carry 1. Show your carries. If the result needs a ninth bit, that is an overflow error.
- Binary shifts: a left shift of n places multiplies by 2^n and a right shift of n places divides by 2^n . Bits that move off the end are lost, so a right shift can lose precision and a left shift can cause overflow.
- Units: 4 bits = 1 nibble, 8 bits = 1 byte, 1000 bytes = 1 KB, then 1000 KB = 1 MB, and so on through GB, TB and PB. Using 1024 is also accepted unless the question says otherwise, but 1000 is much easier without a calculator.
- File sizes in bits: image = colour depth $\times$ width $\times$ height (in pixels); sound = sample rate (Hz) $\times$ bit depth $\times$ duration (s); text = bits per character $\times$ number of characters. Divide by 8 for bytes, then by 1000 for each step up.
- n bits give 2^n different values, so 8 bits allow 256 colours or characters. To find how many bits are needed, find the smallest n where 2^n is at least the number required.
- Give your answer in the unit asked for, and write the unit. If you simplify (for example dividing by 8 early), write each step so the examiner can follow it.
- Check conversions by converting back: if 94 became 01011110, add up $64 + 16 + 8 + 4 + 2$ to get 94 again.

Diagrams and truth tables

- Logic gates: draw the standard symbols clearly. AND has a flat back and a rounded front; OR has a curved back and a pointed front; NOT is a triangle with a small circle at its output. Label every input and the output with the letters from the question.
- Build a logic diagram from the inside of the brackets outwards: for $P = \text{NOT} (A \text{ AND } B)$, draw the AND gate first, then feed its output into the NOT gate.
- Truth tables: list the inputs in binary counting order (00, 01, 10, 11 for two inputs; 000 to 111, eight rows, for three). Add a column for each part of the expression, then the final output, and check every row.
- AND outputs 1 only when both inputs are 1; OR outputs 1 when at least one input is 1; NOT reverses its input.
- Network diagrams: in a star topology every device connects to a central switch; in a mesh topology devices connect directly to each other, giving many routes (in a full mesh, every device connects to every other). Label the devices if asked.
- Flowcharts: use the right symbols (rounded box for start and stop, parallelogram for input and output, rectangle for a process, diamond for a decision with both exits labelled Yes and No, rectangle with double side lines for a sub program) joined by arrows.
- Structure diagrams: break the problem into sub-tasks in levels from the top down; each box is a smaller part of the box above it.
- Use a ruler and pencil for diagrams, and draw them big enough to read easily.

Trace tables, searching and sorting

- Trace tables: use one column per variable and one for output. Work through the code line by line and write a new value only when it changes. Don't guess the pattern; questions are often written so that the pattern changes part way through.
- Watch the loop limits: in ERL for `i = 1 to 5` runs with `i = 1, 2, 3, 4` and `5`. A while loop checks its condition before each repeat; a `do ... until` loop always runs at least once.
- Work out MOD (the remainder) and DIV (whole-number division) carefully: $17 \text{ MOD } 5 = 2$ and $17 \text{ DIV } 5 = 3$.
- Linear search: check each item in turn from the start until you find the target or reach the end. It works on unsorted data.
- Binary search: the data must be sorted. Look at the middle item, compare it with the target, discard the half that can't contain the target, and repeat. Show the middle item you compared at each step. With an even number of items, choose one rule for the middle (for example $(\text{first} + \text{last}) \text{ DIV } 2$) and use it every time.
- Bubble sort: compare each adjacent pair and swap them if they are in the wrong order; one run through the list is a pass. Show the list after each pass (or each swap, if asked). It stops after a pass with no swaps.
- Merge sort: split the list in half again and again until every item is on its own, then merge the lists back together in pairs, in order. Show every splitting and merging stage.
- Insertion sort: take each item in turn from the unsorted part and insert it into the correct place in the sorted part. Show the list after each insertion.
- Know when each is suitable: binary search needs far fewer comparisons on a large sorted list; merge sort is usually quicker than bubble or insertion sort on large lists but needs more memory; insertion sort works well on small or nearly sorted lists; bubble sort is simple but slow on large lists.

Writing algorithms and code

- In Section A you can usually answer an algorithm question in any clear form (ERL, a high-level language such as Python, pseudocode or a flowchart) unless the question asks for a particular one. In Section B, algorithms must be written in ERL or a high-level programming language you have studied.
- Marks are given for each correct step of logic, such as 'takes an input and stores it', 'loop repeats the right number of times', 'correct condition' and 'outputs the result'. A partly correct program still scores, so always write something.
- Use the exact variable, array and sub program names given in the question, and keep to one language for the whole answer.
- Indent every block and close it (endif, endwhile, next i, endfunction) so the structure can't be misread.
- Initialise totals and counters before a loop (e.g. `total = 0`) and update them inside it.
- Use `==` to compare and `=` to assign, and check whether a limit needs `<` or `<=`. Off-by-one errors and wrong comparisons are among the commonest ways to lose marks in code.
- Choose the right loop: `for` when you know how many times to repeat; `while` or `do ... until` when you repeat until something happens, such as a valid input.
- `input()` gives a string, so cast it with `int()` or `float()` before doing arithmetic, and use `str()` before joining a number onto a string with `+`.
- A function returns a value with `return`; a procedure does not. If asked to write a function, take the data in as parameters and return the result, rather than printing it, unless the question says otherwise.
- SQL: `SELECT` the fields, `FROM` the table, `WHERE` the condition, e.g. `SELECT Name, Form FROM Pupils WHERE Year = 10 AND House = 'Blue'`. Use `*` to select every field.
- Here is the kind of structure mark schemes reward. Task: write a function that returns the mean of the five scores in the array `scores`.

```
function average(scores)
    total = 0
    for i = 0 to 4
        total = total + scores[i]
    next i
    return total / 5
endfunction
```

Each line is a step that can earn a mark: the total starts at 0, the loop visits all five items (index 0 to 4), each item is added, and the mean is returned.

Paper 2 Section B: design, write, test and refine

- Section B has the longer questions where you write, test and refine algorithms and programs, often for a situation such as a booking system, a quiz or a game.
- Read the question and any code you are given carefully before you start. Underline the inputs, the rules (e.g. 'a score must be from 0 to 100') and what must be output or stored.
- In Section B, algorithms must be written in OCR Exam Reference Language or a high-level programming language you have studied. Don't use a flowchart or informal pseudocode for these questions.
- Reuse the identifiers already given in the question (array names, function names, file names). This shows how your code fits with the rest of the program.
- Break a long question into steps and write each one as a few lines: input, validate, process, output. Each step is usually worth marks on its own.
- If you can't write the whole program, write the parts you can. A loop with the right condition, or a correct input and output, still earns marks.
- Typical tasks: validate an input with a loop, total or average the values in an array, find the highest value, count items that meet a condition, write a function with parameters and a return value, read from or write to a file, slice or join strings, and complete a test plan.
- When asked to refine code, keep what already works and change only what the question asks for (for example adding validation, or replacing repeated lines with a loop). Make it clear where your new lines go.
- Before moving on, trace your code in your head (or on the question paper) with one normal value and one boundary value.

Testing, errors and robust programs

- A syntax error breaks the rules of the language, so the program can't be translated and run (e.g. a missing `endif` or a misspelt keyword). A logic error lets the program run but gives the wrong result (e.g. `>` where `>=` was needed).
- When asked to find and fix an error, give the line number, say what is wrong, and write the corrected line in full.
- Test data: normal (valid, typical values), boundary (valid values at the very edge of the range), invalid (the right data type but outside the range) and erroneous (the wrong data type). For 'a whole number from 1 to 10': normal 5, boundary 1 or 10, invalid 11, erroneous 'seven'.
- Expected results must be specific: not 'it works' or 'error', but 'accepted and stored' or 'error message shown and the user is asked to enter it again'.
- Iterative testing happens during development, testing each part as it is written. Final (terminal) testing checks the whole program at the end against the requirements.
- Defensive design: anticipate misuse, authenticate users (e.g. usernames and passwords), validate inputs (range, type, length, presence and format checks), and keep code maintainable with comments, indentation, meaningful identifiers and sub programs. Name the technique and say what it prevents.
- Validation checks that input is sensible and follows the rules; it can't prove the data is correct. A valid but wrong date of birth still gets through.

Checking, getting stuck, extra space and crossing out

- With time left, check in this order: questions you skipped, then any code you wrote (trace it with a test value), then calculations (convert back), then that every question has an answer.
- Stuck? Re-read the question and underline the key words, work out which topic it comes from, and write down what the key terms mean. This often earns a mark and unlocks the rest.
- Never leave code questions blank. An input line, a loop with the right condition or an output line can each be worth a mark.
- If you run out of room, use the additional answer space at the back of the booklet. Write the question number clearly in the margin next to the extra answer, and write 'continued at the back' under the first part.
- Keep your writing inside the answer areas: scripts are scanned for marking, and writing squeezed into the edges may not be seen.
- To cross out, draw one neat line through the work you don't want marked. Never leave two different answers to the same question; the examiner won't choose the better one for you.

Learning from your mocks

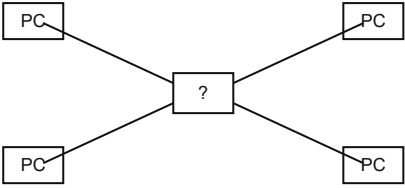
- Mark every mock strictly with the mark scheme, then sort each lost mark by cause: didn't know it, too vague, misread the question, ran out of time, or a careless slip.
- Knowledge gaps: relearn the subtopic from its notes, then do a short practice set on it within a few days.
- Vague answers: rewrite them using the technical terms from the mark scheme, and add a reason to every explain point.
- Code questions: rewrite your program correctly by hand, then type it into an IDE and test it with normal, boundary, invalid and erroneous data.
- Timing: write down the time you reached Section B or the extended response question, and set a target for the next mock.
- A week later, redo every question you dropped marks on without looking at your first attempt.

Command words

Word	What it means	How to answer	Example
State	Give a short, clear fact or name. No explanation is needed.	One precise point per mark, using the correct technical term. A word or short phrase is often enough.	State one purpose of the accumulator. Answer: it stores the result of a calculation carried out by the ALU.
Identify	Pick out or name something, often from code, a diagram or information given in the question.	Point to the exact item: the line number, variable, device or data type. Don't explain unless asked.	Identify the most suitable data type to store a pupil's height in metres, such as 1.62. Answer: real (float).
Give	Produce an answer from what you have learnt or from the information given,	Give exactly the number asked for; make each one different and specific.	Give two examples of embedded systems found in a kitchen. Answer: the control system of a microwave oven; the controller in a dishwasher.

Word	What it means	How to answer	Example
	often an example or a reason.		
Define	Say exactly what a term means.	One accurate sentence in technical language. An example can support a definition but can't replace it.	Define decomposition. Answer: breaking a problem down into smaller sub-problems that are each easier to solve.
Outline	Set out the main points or stages briefly.	Give the key steps or features in order, without the full detail of a description.	Outline how a binary search finds a value. Answer: check the middle item of the sorted list; if it is not the target, discard the half that can't contain it; repeat until the target is found or no items are left.
Describe	Say what something is like, what it does, or the steps of a process in order. Reasons are not needed.	One mark per relevant point or step. Use technical terms and keep the steps in the right order.	Describe how an instruction is fetched. Answer: the address in the program counter is copied to the MAR (1); the instruction at that address is copied from memory into the MDR (1); the program counter is incremented (1).
Explain	Give reasons: why or how something happens.	Make a point, then expand it with 'because', 'so' or 'which means'. Usually one mark for the point and one for the reason.	Explain why a larger cache can improve CPU performance. Answer: more frequently used data and instructions can be held in cache (1), which is faster to access than RAM, so the CPU spends less time waiting for data (1).
Compare	Give similarities or differences between two or more things.	Mention both things in every point and use comparison words such as 'whereas', 'more' and 'less'.	Compare a compiler with an interpreter. Answer: a compiler translates the whole program in one go, whereas an interpreter translates and runs it one line at a time (1); a compiled program can run without the translator, whereas an interpreted program needs the interpreter every time it runs (1).
Discuss	Present the key points about an issue and weigh up different sides, in an extended answer.	Often used for the extended response question. Cover several angles or stakeholders, develop each point with reasons and consequences, apply everything to the scenario, and end with a conclusion.	Discuss the legal, ethical and privacy issues of a supermarket using a loyalty app that records every item its customers buy.
Evaluate	Judge how good or suitable something is by weighing up its	Give points for and against, linked to the scenario, then a clear judgement with a reason. Without a judgement you	Evaluate whether a small hairdresser should store its customer records in the cloud or on a computer in the salon.

Word	What it means	How to answer	Example																				
	strengths and weaknesses.	can't reach the top marks.																					
Justify	Give reasons that support a choice, decision or answer.	Link each reason to details in the scenario. A general advantage that ignores the scenario is not enough.	A photographer carries her laptop to outdoor shoots. Justify her choice of a solid state drive. Answer: it has no moving parts, so it is less likely to be damaged if the laptop is knocked or dropped (1); it is small and light, so the laptop is easy to carry (1).																				
Calculate	Work out a numerical answer.	Write the formula, put in the numbers, show each step, and give the answer in the unit asked for.	Calculate the file size in bytes of an image 20 pixels wide and 10 pixels high with a colour depth of 4 bits. Answer: $20 \times 10 \times 4 = 800$ bits (1); $800 \div 8 = 100$ bytes (1).																				
Convert	Change a value from one number base or unit to another.	Use place values or nibbles, show your working, and give the number of bits asked for, including leading zeros.	Convert the denary number 94 into 8-bit binary. Answer: $64 + 16 + 8 + 4 + 2 = 94$, so 01011110.																				
Show	Give the working or stages that lead to an answer, so the examiner can follow your method.	Write out every stage, such as each pass of a sort or each line of a calculation. A final answer on its own may not score.	Show the first pass of a bubble sort on the list 5, 2, 4, 1. Answer: 2, 5, 4, 1 then 2, 4, 5, 1 then 2, 4, 1, 5.																				
Complete	Fill in the missing parts of a table, diagram, algorithm or program.	Fill in every gap, keep to the format already started, and check that your additions work with what is given.	Complete the truth table for $P = A \text{ AND NOT } B$. Answer: <table border="1" data-bbox="868 1341 1163 1688"> <thead> <tr> <th>A</th><th>B</th><th>NOT B</th><th>P</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td><td>0</td></tr> </tbody> </table>	A	B	NOT B	P	0	0	1	0	0	1	0	0	1	0	1	1	1	1	0	0
A	B	NOT B	P																				
0	0	1	0																				
0	1	0	0																				
1	0	1	1																				
1	1	0	0																				
Draw	Produce a diagram, such as a logic diagram, a network topology or a flowchart.	Use the standard symbols, label inputs, outputs and devices, and draw neatly in pencil with a ruler.	Draw a logic diagram for $Q = \text{NOT } (A \text{ OR } B)$. Answer: an OR gate with inputs A and B, whose output goes into a NOT gate with output Q.																				

Word	What it means	How to answer	Example
Label	Add names or short notes to parts of a diagram.	Write each label next to the right part, with a line pointing to it if there could be any doubt.	Label the device in the centre of this star network.  Answer: write 'switch' next to the central box, with a line pointing to it.
Tick	Choose the correct answer or answers from the boxes given.	Tick exactly the number of boxes asked for. To change an answer, cross the wrong tick out clearly.	Tick one box to show the protocol used to send an email from a device to a mail server: FTP, HTTP, IMAP, SMTP. Answer: SMTP.
Write	Produce an algorithm, program code or an SQL statement.	Use the form the question asks for (in Section B, ERL or a high-level language), the identifiers given, and one clear step per line. Each correct step of logic can earn a mark.	Write an algorithm that asks for a password and keeps asking until it is at least 8 characters long. Answer: <pre>password = input("Enter a password") while password.length < 8 print("Too short, try again") password = input("Enter a password") endwhile print("Password accepted")</pre>
Refine	Improve an existing algorithm or program so it is more robust, more efficient or meets a new requirement.	Keep what already works, change only what is needed, and make it clear where your new or changed lines go.	Refine <code>age = int(input("Enter age"))</code> so that only ages from 11 to 18 are accepted. Answer: after it, add <code>while age < 11 OR age > 18</code>, then <code>age = int(input("Enter an age from 11 to 18"))</code>, then <code>endwhile</code>.

Free notes, worksheets, practice papers and mock exams at chhetriacademy.com. Written for this site; not taken from past papers.
 Stuck? 1-to-1 tutoring with a free 30-minute trial lesson.