

Exam technique

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

How the exams work

Two exams, each worth half of the GCSE, with no coursework. The course is not tiered, so everyone sits the same papers and can get any grade from 1 to 9. Paper 1 is a written theory paper on Topics 1 to 5. Paper 2 is taken on screen: you read, fix, complete and write Python 3 programs in an IDE, using the coding files provided and Pearson's Programming Language Subset (PLS).

Paper	Time	Marks	Calculator	What's on it
Paper 1: Principles of Computer Science	1 h 30 min	75	Not allowed	Topics 1 to 5: computational thinking (decomposition, abstraction, algorithms, searching and sorting, truth tables), data (binary, two's complement, shifts, hexadecimal, characters, images, sound, file sizes, compression), computers (hardware, software, programming languages and translators), networks (including network security) and issues and impact (environmental, ethical and legal issues, cybersecurity). Written on the paper: multiple choice, short answers, calculations, algorithms and trace tables, and extended Discuss answers marked by level.
Paper 2: Application of Computational Thinking	2 h	75	Not needed: the paper is taken on screen in a Python 3 IDE	Topic 6: problem solving with programming, using the algorithm skills from Topic 1. You work in the coding files provided: fixing errors, completing and amending programs, and writing programs from a set of requirements, with the Programming Language Subset (PLS) to refer to. Your answers are the Python code you save in those files.

Exam technique

Know how the two papers differ

- Paper 1 tests what you know about Topics 1 to 5 and whether you can apply it to a short scenario. It also asks you to trace, complete and write algorithms, fill in truth tables and do binary and file size working by hand.
- Paper 2 is practical. Every mark comes from Python code in the files you save. You can run your code during the exam, so you are expected to test it and hand in programs that work.
- Both papers are out of 75, but Paper 2 gives you 30 minutes more, because writing, running and fixing code takes longer than writing an answer.
- Revise them differently. Paper 1 rewards exact terms and answers linked to the context. Paper 2 rewards hours spent actually programming at a keyboard, not reading about programming.
- The papers overlap through Topic 1: the algorithms you trace on paper in Paper 1 (totals, counts, validation loops, linear and binary search, bubble sort) are the kind you may have to code in Paper 2.

Preparing in the weeks before

- List every subtopic in Topics 1 to 6 and rate yourself red, amber or green. Spend most of your time on red and amber, and re-rate every week.
- For Paper 1, learn definitions in precise words (abstraction, embedded system, protocol, lossless compression), processes in order (the fetch–decode–execute cycle), and advantages and disadvantages in pairs (wired and wireless, compiler and interpreter, lossy and lossless).
- Drill the by-hand skills until they are automatic: binary to denary and back, two's complement, binary addition and overflow, logical and arithmetic shifts, binary to hexadecimal and back, and file size and transmission time expressions.
- For Paper 2, program in Python on most days, even for 20 minutes. Type programs from scratch: in the exam there is no internet and nothing to copy from.
- Get to know the PLS so well that you can find a string method, a file-handling line or a turtle command in seconds. Stick to what it shows so the exam feels familiar.
- Practise in a plain IDE like the one your school will use in the exam. Learn how to run a file, read an error message and go to the line it points to.
- Do at least two full timed papers of each kind before the real exams, and mark them strictly against the mark scheme.

The night before and the morning of the exam

- Before Paper 1: read through your cheatsheets (powers of 2, the data units, key definitions, protocols and the four layers of the TCP/IP model), then stop. Sleep will earn you more marks than one more hour of cramming.
- Before Paper 2: write two or three short programs from memory, for example a validation loop, a function that returns a value and a loop that reads a file line by line. This warms you up; it is not the time to learn anything new.
- Pack black pens, a pencil, a ruler and a rubber for Paper 1, for flowcharts, truth tables and diagrams. You cannot use a calculator.
- In the morning, eat breakfast, arrive early, and run through the powers of 2 up to $2^{10} = 1024$ and the unit ladder (bit, nibble, byte, kibibyte, mebibyte, gibibyte, tebibyte) in your head.
- For Paper 2, rehearse the routine: open the file the question names, work in it, save it under the name the question asks for, run it, save again.

The first five minutes

- Paper 1: fill in the front cover, then look through the whole paper. Note what each question is about and where the extended Discuss answers are, so you can protect time for them.
- Paper 1: jot the place values 128, 64, 32, 16, 8, 4, 2, 1 on a blank part of the paper so every binary conversion starts from the same reliable line.
- Paper 2: check that you can see the folder of coding files and the PLS, and that you can open and run a file in the IDE. Put your hand up straight away if anything will not open.
- Paper 2: read every question before you start, noting the marks. The later, longer programming tasks carry the most marks, so decide now roughly what time you need to reach them.

Timing

- Paper 1: 90 minutes for 75 marks is about 1.2 minutes per mark. An extended Discuss answer gets about 1.2 minutes per mark (about 7 minutes if it is worth 6), a 2-mark Explain about 2 to 3 minutes, and a 1-mark State well under a minute.
- Paper 1 is made up of several multi-part questions. Look at the marks for each one, work out roughly when you should finish it, and check the clock at the end of each one.
- Aim to finish Paper 1 with about 5 minutes left to check your calculations, your multiple-choice crosses and any parts you skipped.
- Paper 2: 120 minutes for 75 marks is about 1.6 minutes per mark, so a 10-mark task gets about 16 minutes, including testing.
- In Paper 2, give yourself a time to move on. If a bug has beaten you for 5 minutes, save the file, note the question number and come back later with fresh eyes.
- Keep the last 10 minutes of Paper 2 for running every file you have changed one last time and checking each is saved with the right name.
- If you finish early, add a further developed point to your Discuss answers in Paper 1, and in Paper 2 test your programs with boundary and invalid data.

Reading the question

- Find the command word first, then the marks. 'State' for 1 mark wants a term or short phrase; 'Explain' for 2 marks wants a point and a reason linked to it.
- Count what is asked for: 'give two', 'one advantage', 'one method'. Extra answers earn nothing, so write your best ones first and stop.
- Underline the context: a school network, a smart thermostat, a delivery company. Answers that could be about any system usually earn fewer marks than answers that use the scenario.
- Check the form the answer must take: 'in kibibytes', 'as an 8-bit binary number', 'in hexadecimal', 'using two's complement'. An answer in the wrong form or unit can lose marks.
- When a question says you do not need to do the calculation, the expression itself is the answer, so make sure every number and every conversion is in it.
- In Paper 2, treat the list of requirements as a checklist: each one is usually worth marks. Where the question gives exact prompts, messages or an output layout, use them exactly.

Multiple-choice and short-answer questions

- Pearson multiple choice: put a cross in the box. If you change your mind, put a line through that box and cross the new one. Never leave one blank.
- Rule out options that are clearly wrong first. Often two options are nearly right, so test each against the exact definition.
- 'Give', 'State' and 'Name' need one precise item per mark. Use the technical term: 'program counter', 'router', 'lossless', not 'the thing that stores it'.
- 'Identify' asks you to pick something out, for example a line number, a data type, a vulnerability in a scenario or the item a search checks next.
- 'Define' needs an accurate one-sentence meaning. Learn the key definitions so you can write them without thinking.
- In 2 and 3 mark answers, each mark needs a different point. Two sentences that say the same thing in different words get one mark.

Describe and explain questions

- 'Describe' means say what something is, does or how it happens, as features or steps. You do not need reasons. One accurate point per mark.
- 'Explain' means give reasons. Make the point, then develop it with 'because', 'so' or 'which means': 'A mesh topology is more reliable (1) because data can take another route if one link fails (1).'
- Link every point to the scenario: 'the shop's card readers keep working' beats 'devices keep working'.
- Words like 'faster', 'cheaper', 'easier', 'safer' and 'more efficient' score nothing on their own. Say faster at what, and why.
- For advantages and disadvantages, make the comparison clear: 'uses less storage than the original file' rather than 'uses less storage'.

Extended Discuss answers

- Paper 1 includes extended answers, usually with the command word 'Discuss'. They are marked by level: the examiner reads the whole answer and judges your knowledge and how well you apply it and weigh it up.
- They often come from issues and impact (environmental, ethical and legal issues, cybersecurity), or ask you to weigh up a choice of technology in a given situation.
- Plan for 30 seconds: jot three or four points. If the question names aspects, such as ethical, legal and environmental, cover every one of them.
- Write developed points: point, explanation, link to the scenario. Three well-developed points are worth more than six one-word ideas.
- Show more than one side: benefits and drawbacks, or who gains and who loses. You do not need a final verdict, but a closing sentence that weighs things up shows balance.
- Use the specification's vocabulary: data protection legislation, consent, e-waste, algorithmic bias, open-source and proprietary licences, penetration testing.

Calculations without a calculator

- Know the powers of 2 up to 2^{10} and the units: 4 bits = 1 nibble, 8 bits = 1 byte, 1024 bytes = 1 kibibyte (KiB), 1024 KiB = 1 MiB, 1024 MiB = 1 GiB, 1024 GiB = 1 TiB.
- Image file size in bits = width in pixels × height in pixels × colour depth. Sound file size in bits = sample rate × bit depth × length in seconds (× the number of channels if the question gives more than one). Divide by 8 for bytes, then by 1024 for each step up.
- Transmission time in seconds = file size ÷ transmission rate. Put both in the same unit first: rates are in bits per second, so a file size in bytes must be multiplied by 8.
- 'Construct an expression' questions: write the whole calculation with the right numbers, operations and conversions, e.g. $(800 \times 600 \times 24) \div 8 \div 1024$ for an image size in KiB.
- Show your working in every multi-step calculation or conversion. Method marks can still be awarded when the final number is wrong.
- Binary answers must have exactly the number of bits asked for, with leading zeros (8-bit means 8 digits). In two's complement, check that the leftmost bit gives the sign you expect.
- Check conversions by going back the other way: turn your binary answer back into denary, or your hexadecimal back into binary.

Algorithms, trace tables and truth tables in Paper 1

- Algorithms in questions can be flowcharts, pseudocode or Python code, so you need to read all three fluently.
- Trace tables: work one line at a time, write a value only when a variable changes, check the loop condition every time round, and write outputs exactly as they would appear.
- Completing an algorithm: read all of it first to see what each variable does, then fill each gap so it fits the variable names and structure already there.
- Writing an algorithm: use a flowchart, pseudocode or program code unless the question names one. The marks are mostly for correct logic: inputs, the right loop, the right conditions, the outputs.
- Errors: give the line number, the type of error (syntax, logic or runtime) and the corrected code if asked.
- Searches and sorts: show every step the question asks for, such as each pass of a bubble sort, each split and merge in a merge sort, or each middle item checked in a binary search.
- Truth tables: list every combination of inputs in order (three inputs give 8 rows, from 000 to 111), and work out any middle columns, such as A AND B, before the final output.

Paper 2: working on screen

- Open the file the question names, make your changes in it and save it with exactly the file name the question asks for. Work that is not saved, or is saved somewhere else, may not be marked.
- Save every few minutes. Do not rely on the IDE to save for you.
- Run your code often. A syntax error stops the whole file running, so fix it before you add anything else.
- Keep the PLS open beside your code. It shows the Python you are expected to use, so check it rather than relying on half-remembered Python.
- Keep the code you are given, its variable names and its layout unless the question tells you to change them. The question is built around that code.
- When a question asks for comments, write ones that explain the purpose of a section ('checks the PIN is 4 digits'), not ones that repeat the code.
- Never leave a file untouched. A program that only partly works can still earn marks for the parts that are right.

Paper 2: the kinds of programming task

- Fix the errors: run the program to find syntax and runtime errors from the error messages, then test it with data where you know the right answer to find logic errors.
- Complete the program: fill in the missing parts, using the variables and structures that already exist.
- Put code in order: where lines are given in the wrong order, rebuild the program and keep the indentation that shows which lines belong inside each loop or if.
- Amend or extend: add a feature, such as validation, a menu option or an extra output, without breaking what already works. Test the old features again afterwards.
- Write a program from requirements: break it down into inputs, processing and outputs, then write and test one requirement at a time, starting with the ones you are sure of.
- Skills that come up: string handling, lists and two-dimensional lists, reading and writing text files (often comma-separated), random numbers, functions with parameters and return values, validation, and turtle graphics.
- In the longer tasks, readability counts: meaningful names, constants in capitals, comments, indentation and white space.
- A reliable pattern for a range check that repeats until the input is valid:

```
age = int(input("Age: "))
while age < 11 or age > 18:
    print("Enter an age from 11 to 18")
    age = int(input("Age: "))
```

Checking and testing

- Paper 1: re-read each answer against its command word and marks. Does an Explain give a reason? Does a Give two give two different things?
- Paper 1: check units and conversions: bits or bytes, 1024 not 1000 between KiB, MiB and GiB, and the number of bits in every binary answer.
- Paper 1: check every multiple-choice question has exactly one cross, and that nothing is left blank.
- Paper 2: test every program with normal, boundary and erroneous data. For a valid range of 1 to 100, try 50, then 1 and 100, then 0 and 101.
- Paper 2: look for off-by-one errors. `range(1, 10)` gives 1 to 9, not 1 to 10.
- Paper 2: check the output matches the question exactly: the messages, the order, and any rounding, e.g. `round(total, 2)`.
- Paper 2: remove or comment out any extra `print()` lines you added while debugging, so the output is only what the question asks for.

When you are stuck, extra space and crossing out

- Paper 1: write something for every question. A correct key term or the first step of a method can earn a mark. Then move on and come back later.
- An algorithm you cannot finish still earns marks for its parts: write the inputs, the loop and the output you are sure of.
- Paper 2: if one feature will not work, make everything else work first. Print the values of variables to see where the program goes wrong.
- Paper 2: if a whole task stumps you, write what you can: take the inputs, set up the loop, output something. Marks are given for each part that is right.
- If you run out of space in Paper 1, carry on in the extra space provided or on extra paper, and write 'continued on page ...' in the original answer space with the question number.
- To remove work, draw one neat line through it. Do not leave two different answers for the same part, or you may lose the mark.

Learning from mocks

- Mark each mock with the mark scheme and write down why each mark went: didn't know it, misread the question, too vague, ran out of time, or a code error.
- Keep an error log sorted by topic, and re-test yourself on those topics a week later.
- Take Paper 2 practice at a computer, in an IDE, with no internet and only the PLS, and time it.
- Compare your answers with the mark scheme's wording and note the key words that earned the marks. Use them next time.
- Redo every question you dropped marks on until you can get full marks without help.

Command words

Word	What it means	How to answer	Example
Give / State / Name	Recall a fact, term or short answer. No explanation is needed.	A word, phrase or short sentence for each mark. Use the technical term.	State the name of the CPU register that holds the address of the next instruction. <i>Program counter (1).</i>
Identify	Pick out or recognise something, often from information, code or a scenario you have been given.	Select exactly what is asked for, such as a line number, a variable or a threat. Add no explanation unless asked.	In a small office, the router still has the administrator password it came with from the factory, and all staff have strong passwords of their own. Identify one vulnerability in this network. <i>The router still uses its default administrator password (1).</i>
Define	Give the precise meaning of a term.	One accurate sentence using the right technical words. Learn the key definitions exactly.	Define the term 'abstraction'. <i>Removing unnecessary detail from a problem so you can focus on the parts that matter for solving it (1).</i>
Describe	Give an account of what something is, does or how a process happens. No reasons are needed.	One accurate feature or step per mark. For a process, give the steps in order.	Describe how a bubble sort puts a list into ascending order. <i>Compares each pair of adjacent items (1); swaps them if they are in the wrong order (1); repeats passes until a pass makes no swaps (1).</i>

Word	What it means	How to answer	Example																								
Explain	Give reasons: show how or why something happens or is the case.	Make a point, then develop it with 'because', 'so' or 'which means', linked to the context. Usually a point and its development for each 2 marks.	Explain one reason why a video streaming service uses lossy compression. <i>It makes the files much smaller (1), so the video can be sent over the internet quickly enough to play without pausing (1).</i>																								
Discuss	Explore an issue or situation, looking at different sides and what the consequences are.	Several developed points covering every aspect the question names, with both benefits and drawbacks, linked to the scenario. Marked by level.	A council plans to replace all its staff laptops every two years. Discuss the environmental and ethical issues of this plan. (6)																								
Complete	Fill in the missing parts of a table, diagram, algorithm or program.	Use the information given, keep to the style and names already used, and fill every gap.	Complete a trace table for this algorithm when the input is 3. <pre> n = int(input()) total = 0 for i in range(1, n + 1): total = total + i print(total) </pre> Answer: <table> <tr> <th>n</th><th>i</th><th>total</th><th>output</th></tr> <tr> <td>3</td><td></td><td>0</td><td></td></tr> <tr> <td></td><td>1</td><td>1</td><td></td></tr> <tr> <td></td><td>2</td><td>3</td><td></td></tr> <tr> <td></td><td>3</td><td>6</td><td></td></tr> <tr> <td></td><td></td><td></td><td>6</td></tr> </table>	n	i	total	output	3		0			1	1			2	3			3	6					6
n	i	total	output																								
3		0																									
	1	1																									
	2	3																									
	3	6																									
			6																								
Calculate	Work out a numerical answer.	Show each step, convert units first, and give the unit asked for in your answer.	Calculate the size, in bytes, of a 16×16 pixel image with a colour depth of 4 bits. $16 \times 16 \times 4 = 1024 \text{ bits (1)}$; $1024 \div 8 = 128 \text{ bytes (1)}$.																								
Construct	Build something from the information given, most often an expression for a calculation.	For an expression, write the full calculation with the correct values, operations and unit conversions. You do not have to work it out unless the question asks.	Construct an expression to calculate the time, in seconds, to send a 5 MiB file at 20 000 000 bits per second. $(5 \times 1024 \times 1024 \times 8) \div 20\,000\,000$																								

Word	What it means	How to answer	Example
Convert	Change a value from one number base or unit to another.	Show your working (place values, or groups of 4 bits for hexadecimal) and give exactly the number of bits or digits asked for.	Convert the binary number 1011 0110 to hexadecimal. <i>1011 = B, 0110 = 6, so B6 (2).</i>
Draw	Produce a diagram, such as a flowchart or a network topology.	Use the standard symbols, draw lines with a ruler, show the direction of flow and label anything that is not obvious.	Draw a flowchart for an algorithm that keeps asking for a password until the user enters the correct one, then outputs 'Welcome'.
Add / Label	Put extra information or labels onto a diagram, table or program you have been given.	Place each label on or clearly next to the right part, and add only what is asked for.	Add a comment to this line of code to explain what it does: <code>total = total + score.</code> <i># adds this score to the running total (1)</i>
Write	Produce an algorithm (Paper 1) or a program (Paper 2) that meets the requirements given.	Cover every requirement: inputs, processing and outputs. Use meaningful names and correct indentation, and in Paper 2 run and test it.	Write a program that asks for ten test scores and then displays the highest score and the mean score.
Amend	Change existing code so it works correctly or does something extra.	Keep what already works, change only what is needed, save under the name given and test the whole program again.	Amend this code so that the user has three attempts to enter the correct PIN before the program displays 'Card locked' and ends. <pre>pin = input("PIN: ") if pin == "1234": print("Welcome") else: print("Card locked")</pre>

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.