Computational thinking

2.1.1 · Algorithms and programming (Paper 2) · revision notes

Computational thinking is the way you break down and simplify a problem so that a computer can solve it. It has three parts: abstraction, decomposition and algorithmic thinking. Expect short definitions and questions that ask you to apply the terms to a scenario.

Key facts

- Abstraction = removing unnecessary detail to focus on what matters
- Algorithmic thinking = working out the steps, in order, that solve the problem
- Benefits of decomposition: easier to code and test, shared between a team, parts reused
- Decomposition = breaking a problem into smaller sub-problems
- Algorithm = a sequence of steps that solves a problem
- Benefits of abstraction: a simpler problem and less to program

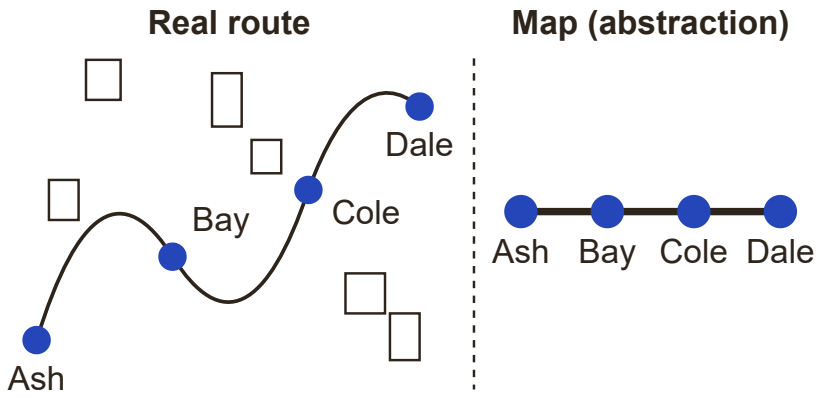
Notes

The three parts of computational thinking

- **Abstraction**: removing or hiding details that are not needed, so you can focus on what matters for the problem.
- **Decomposition**: breaking a complex problem into smaller sub-problems. Each sub-problem can be broken down again until every part is simple enough to solve.
- **Algorithmic thinking**: identifying the steps needed to solve the problem and putting them in the right order, so the solution can be followed by a computer. It includes spotting where a decision is needed and where steps repeat.

Abstraction in practice

- A map of a train line shows the stations and the order they come in, but leaves out the true distances, the shape of the track and the buildings nearby.

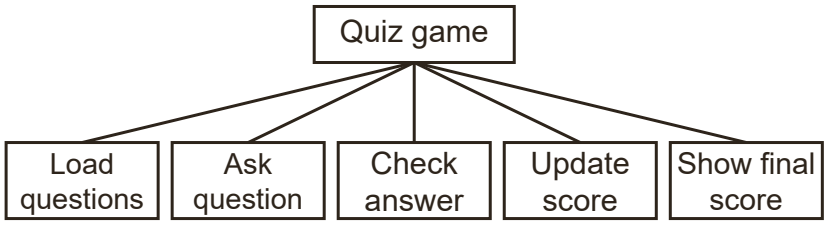


Abstraction: the map keeps only what matters, the stations and their order

- A program that simulates rolling a dice only needs a number from 1 to 6. The colour, size and weight of a real dice are abstracted away.
- To decide whether a detail can be removed, ask: does the program need it to produce the correct result? If not, leave it out.
- Abstraction makes the problem simpler, so the program is easier and quicker to write.

Decomposition in practice

- A quiz game could be decomposed into: load the questions, ask a question, check the answer, update the score, show the final score.



Decomposition of the quiz game into sub-problems

- Each sub-problem is smaller, so it is easier to design, code and test.
- Different programmers can work on different parts at the same time, so the program is finished sooner.
- Parts can be written as sub programs and reused, and errors are easier to find because each part can be tested on its own.
- Decomposition is often shown as a structure diagram.

Algorithmic thinking in practice

- Think about the inputs, the order of the steps, where a decision is needed (selection) and what repeats (iteration).
- Example: to find the highest score in a list, take the first score as the highest so far, compare each of the other scores with it in turn, replace it when a score is higher, then output it at the end.

How to answer each type of question

Define a computational thinking term

1 to 2 marks | Grade 3

1. Give the core idea: removing detail (abstraction), breaking down (decomposition) or working out the steps in order (algorithmic thinking).
2. For a second mark, add what it achieves or a short example.

Example. Define the term **decomposition**. [2]

Model answer. Breaking a problem down into smaller sub-problems (1) so that each part is easier to understand and solve / can be tackled separately (1).

Apply abstraction to a scenario

2 to 4 marks | Grade 5

1. Read what the program has to do.
2. Details it does not need to do that job have been removed (abstracted).
3. Details it needs to do that job are kept.
4. Use the objects named in the scenario, not general answers.

Example. A school is writing a program that simulates a fire drill. It shows students moving from their classrooms to the exits on a plan of the school.
Give **two** details that could be removed by abstraction, and **one** detail that must be kept. [3]

Model answer.

Removed, any two: the colour of the walls / the names of the students / the pictures on display / what the students are wearing (1 each, max 2).

Kept: the positions of the corridors, doors and exits / the number of students in each room (1).

Decompose a problem

3 to 4 marks | Grade 6

1. Read the scenario and list the separate jobs the system must do.
2. Give each sub-problem a short name: a verb and a noun, such as 'calculate the cost'.
3. Make sure each one is a genuine part of the problem, not a restatement of the whole question.

Example. A cinema wants a program that lets customers book tickets online.
Decompose the problem into **four** sub-problems. [4]

Model answer. Any four, 1 mark each: display the films and showing times (1); let the customer choose seats and check they are free (1); calculate the total cost (1); take the payment (1); send a confirmation of the booking (1).

Explain the benefits of decomposition 2 to 4 marks **Grade 7**

1. State a benefit.
2. Explain why it helps, linked to the scenario if there is one.
3. Repeat for the second benefit.

Example. A team of programmers is writing a large program for a bank.
Explain **two** benefits of decomposing the problem. [4]

Model answer.

Each sub-problem is smaller (1) so it is easier to code and test on its own (1).
Different programmers can work on different sub-problems at the same time (1) so the program is completed more quickly (1).
Also accept: parts can be reused (1) as sub programs in other programs (1).

Don't lose marks

- Mixing up abstraction (removing detail) and decomposition (breaking down).
- Defining abstraction as 'making it simple' without saying that unnecessary detail is removed.
- Giving general answers when the question is about a scenario. Use the scenario's objects.
- Listing sub-problems that are vague, such as 'do the program', or that just repeat the question.

More tips

Memory tricks

- Abstraction = Away with detail. Decomposition = Divide it up. Algorithmic thinking = Arrange the steps.
- Detail test for abstraction: 'Would the program give a different result without this?' If not, it can be removed.
- When you see 'decompose', think of a structure diagram: the whole problem at the top, smaller jobs underneath.

Exam technique

- Learn the three definitions word for word: they are quick marks.
- 'Identify' or 'State' needs a short answer; 'Explain' needs a point and a reason ('... so that ...').
- In scenario questions, every point you make should mention something from the scenario.

What each grade needs

What you need to be able to do, from the first marks up to the top grade.

Grade 3 **Define abstraction**
Abstraction is removing unnecessary detail so you can focus on the parts of a problem that matter.

Grade 3 **Define decomposition**
Decomposition is breaking a problem down into smaller sub-problems that are easier to solve.

Grade 4

Define algorithmic thinking

Algorithmic thinking is working out the steps needed to solve a problem and the order they must happen in.

Grade 5

Apply abstraction to a scenario

Say which details a program or model keeps and which it ignores, using the scenario's own objects.

Grade 6

Decompose a given problem into sub-problems

Split a system, such as a booking app or a game, into sensible named tasks.

Grade 7

Explain the benefits of decomposition

Smaller parts are easier to code and test, can be shared between programmers and can be reused.

Free notes, Full packs, 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.