Decomposition and abstraction

1.1 · Computational thinking · revision notes

Decomposition and abstraction are two key ideas of computational thinking: breaking a problem into smaller parts, and leaving out detail that does not matter. This subtopic also covers why programmers use subprograms. Expect definitions and 'give' or 'explain' questions set in an unfamiliar scenario, usually worth 1 to 4 marks each.

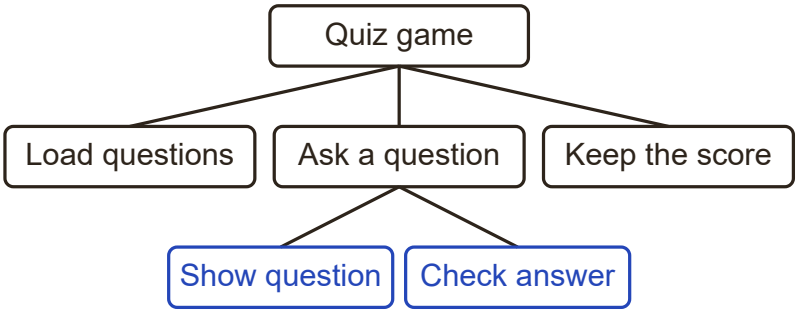
Key facts

- Decomposition: breaking a problem down into smaller, more manageable sub-problems
- A model of something in the real world is an abstraction of it
- Function: returns a value. Procedure: does not return a value
- Decomposition benefits: simpler parts, teamwork, parts tested separately, reuse
- Abstraction: removing unnecessary detail to focus on what matters for the problem
- Subprogram: a named block of code that performs one task and runs when it is called
- Parameter: a value passed into a subprogram when it is called
- Subprogram benefits: no repeated code, easier testing, easier maintenance, reuse, teamwork, readability

Notes

Decomposition

- **Decomposition** means breaking a complex problem down into smaller **sub-problems**. You keep breaking each part down until it is simple enough to solve on its own.

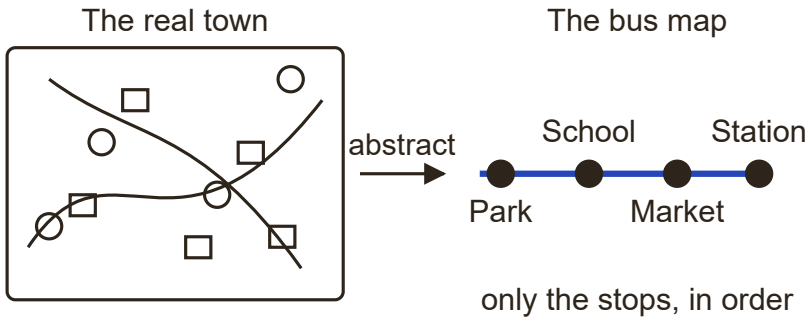


Decomposition: keep breaking each part down until it is simple enough to solve on its own

- Each sub-problem is easier to understand, solve and test than the whole problem.
- Different people can work on different sub-problems at the same time, so the whole solution is finished sooner.
- A sub-problem that comes up more than once, such as 'check that a date is valid', only needs solving once. The solution can then be reused.

Abstraction

- **Abstraction** means removing (or hiding) unnecessary detail so that you can focus on the details that matter for solving the problem.
- Any **model** of the real world is an abstraction. A sat nav's model of a town keeps the roads, junctions and speed limits, but leaves out the colour of the houses.
- Which details matter depends on the **purpose**. A journey-time app needs distances and speed limits; a bus map for passengers only needs the stops and their order.



The bus map is an abstraction: houses, trees and the shape of the roads are left out because passengers do not need them

- Abstraction makes a problem simpler to understand and solve, and the program simpler, quicker to write and less demanding on memory and processing.
- Subprograms use abstraction too: to use `len()` you only need to know what it does, not how it works.

Subprograms

- A **subprogram** is a named, self-contained block of code that carries out one task. It runs whenever the program **calls** it by its name. Each sub-problem from decomposition can become its own subprogram.
- A **function** returns a value to the code that called it; a **procedure** carries out a task but does not return a value. Values are passed in through **parameters**.
- This function is written once but can be called with any values. The call below outputs 12.

```
def area(length, width):
    return length * width

print(area(4, 3))
```

- Benefits: the code is written once and called as many times as needed, so there is less repeated code and fewer chances for errors. Each subprogram can be tested on its own, so errors are easier to find.
- More benefits: a change is made in one place only, so the program is easier to maintain; different programmers can write different subprograms at the same time; subprograms can be reused in other programs; the main program is shorter and easier to read.

How to answer each type of question

Define decomposition or abstraction

1 mark each | Grade 3

1. Give the key idea in one sentence.
2. Decomposition: breaking a problem down into smaller sub-problems.
3. Abstraction: removing **unnecessary** detail. Say that the detail is unnecessary, not just 'removing detail'.

Example. (a) Define the term **decomposition**.
(b) Define the term **abstraction**.

Model answer.

(a) Breaking a problem down into smaller, more manageable sub-problems (1)

(b) Removing unnecessary detail so that only the features that matter for the problem are considered (1)

Explain two benefits of using subprograms

2 to 4 marks | Grade 5

1. Pick two different benefits, not the same idea twice.
2. For each one, add the consequence, joined with 'so'.
3. Use the scenario: name the program or the subprograms in the question.

Example. A team of three programmers is writing a quiz app for a museum. The app will be written as a set of subprograms, such as `askQuestion()` and `updateScore()`.
Explain **two** benefits of writing the app as a set of subprograms.

Model answer.

`askQuestion()` is written once and called for every question (1), so there is less code to write and fewer chances for errors (1). Each programmer can write a different subprogram at the same time (1), so the app is finished sooner (1).

Also accept: each subprogram can be tested on its own, so errors are easier to find; a change is made in one place, so the app is easier to update.

Decompose a scenario into sub-problems

2 to 4 marks | Grade 5

1. Read the scenario and underline each thing the system must do.
2. Write each sub-problem as a task that starts with a verb, e.g. 'calculate the fare'.
3. Give different tasks, not the same task twice in different words.

Example. A leisure centre wants a program that lets members book badminton courts online.
Give **three** sub-problems that the program must solve.

Model answer. Any three from: log in / check the member's details (1); show which courts are free at each time (1); let the member choose a court and a time (1); take payment (1); send a booking confirmation (1); cancel a booking (1). Maximum 3 marks.

Explain how abstraction is used in a model

2 to 4 marks | Grade 6

1. Say which details are kept, and why the purpose needs them.
2. Say which details are left out, and why they do not matter for the purpose.
3. Use details from the scenario, not general ones.

Example. A running app records a runner's position every 10 seconds. It uses the positions to work out the distance run and the average speed.
Explain why the app's model of a run is an abstraction.

Model answer. It keeps only the details it needs to work out the distance and speed: the positions and the times they were recorded (1). It leaves out details that are not needed for these calculations, such as what the runner is wearing, the music they are listening to or the scenery (1).

Don’t lose marks

- Mixing up the two terms: splitting a problem into parts is decomposition, not abstraction.
- Writing 'abstraction removes detail' without saying that the detail is unnecessary or not relevant to the problem.
- Saying that subprograms make a program run faster. That is not a benefit to give: the benefits are about writing, testing, reusing and maintaining code.
- Giving the same benefit twice in different words, e.g. 'easier to read' and 'easier to understand'.
- Giving vague answers in a scenario question. A sub-problem such as 'do the calculations' is too vague; say which calculation.

More tips

Memory tricks

- Decomposition = break it down. Abstraction = leave it out.
- Test for abstraction: does the program need this detail to do its job? If not, it can be left out.
- Subprograms: 'write once, call many times'. Most of the benefits follow from this.
- Subprogram benefits: **RTMS** = Reuse, Test on its own, Maintain in one place, Share the work.

Exam technique

- In 'explain' questions a benefit usually earns one mark and its linked consequence a second, so join them with 'so' or 'because'.
- In scenario questions, use the names, data and subprograms given in the question.
- Write sub-problems as tasks (verb first). Examiners want things the program must do, not pieces of data.
- If you are asked for details a model must include, choose ones that directly affect what the program works out or shows.

What each grade needs

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

Grade 3 — **Define decomposition**
Breaking a problem down into smaller sub-problems that are each easier to understand and solve.

Grade 3 — **Define abstraction**
Removing or hiding unnecessary detail so that you can focus on what matters for the problem.

Grade 4 — **Give benefits of decomposition**
Sub-problems are simpler, can be shared across a team, tested separately and reused.

Grade 5 — **Decompose a scenario into sub-problems**
Break a given system, such as an online booking service, into separate tasks.

Grade 5 — **State the benefits of using subprograms**
Code is written once and called many times, and is easier to test, maintain and share.

Grade 6 — **Apply abstraction to a real-world model**
Say which details a model must keep and which it can leave out, linked to its purpose.

Grade 7 — **Explain benefits in the context given**
Link each benefit to the scenario with a clear consequence, e.g. '... so the app is finished sooner'.

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.