# **Computational thinking**

These ways of thinking help you to solve problems.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aspect | Definition | Meaning | Theoretical positives / Negatives | Simple examples |
| **Thinking abstractly** | Removing unnecessary details and including only the relevant details. | Identifying what does and doesn’t matter to solving the problem.  The idea of layering or levels of a problem.  Deciding what variables & objects will be needed. | + Simplifies the problem / interface.  + Less computation / data.  + Easier to see how the solution to one problem can also be the solution to another.  - Models will not be as accurate. | Icons / symbols on a map.  Charting data.  Moving nodes on a graph data structure to change how it looks. |
| **Thinking ahead** | Identifying the preconditions of a system: inputs, outputs, and reusable components. | What you need to know before you can solve the problem.  The state of data for an algorithm to work.  Identifying what data is required before it is needed (caching)  Identifying reusable program components. | + Caching can speed up a process.  - Caching can be complicated to implement.  - Caching requires the correct data to be fetched for the next instruction. | Working out how much paint you need before starting to decorate.  Getting your debit card out before it is needed to be scanned.  Sorting data for a binary search. |
| **Thinking procedurally** | Breaking a problem down. | Identifying several smaller sub-problems.  Determine the order of events. | + Problems are easier to solve.  + Debugging is easier.  - May not be entirely possible with an event driven rather than procedural approach to programming. | Generating a subject grade requires putting marks into a system, before applying a grade boundary, before printing results. |
| **Thinking logically** | Identifying individual steps and decision points of an algorithm. | Identify the points at which a selection or iteration is needed.  Determine the conditions of the decision.  Determine the next steps depending on the outcome of the decision. | + Makes writing an algorithm easier.  + The complexity of an algorithm can be determined.  + Algorithms can be simplified, or better solutions found more easily.  + Identifies branches for testing. | Happens after thinking procedurally. Using a flowchart or pseudocode to identify the individual steps of an algorithm. |
| **Thinking concurrently** | More than one process happening at the same time. | Identifying parts of the problem can be executed at the same time. | + Increase in speed.  - May be difficult to program.  - Can result in deadlock.  - Problem may not suit concurrency. | When building a house, ordering the windows, while putting up the walls.  Playing sound in a game while taking user inputs.  Multiple images downloading for a webpage. |

# **Computational problems**

A problem is defined as being amenable to a computational approach if there are algorithms that can solve it within a finite number of steps. Problems exist that can theoretically be solved in a finite number of steps but would take far too long to solve with today’s technology, so they are not considered amenable to computation. Cracking 256-bit encryption is an example; it is possible but would take at least 3 x 1052 years. Given infinite time and computing power, you cannot detect all metamorphic malware. The halting problem is a perfect illustration of a problem that is not amenable to a computational approach – a computer can never know if it is stuck in an infinite loop.

Computers:

* Are suited to problems with sequential, logical steps.
* Can perform operations extremely quickly compared to humans.
* Eliminate human error (other than logic errors).
* Can store and retrieve large amounts of data.
* Are efficient at finding patterns in data.

# **Computational methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aspect | Definition | Meaning | Theoretical positives / Negatives | Simple examples |
| **Problem recognition** | Knowing what the problem is. | Identifying the key requirements of a solution. | - Not all problems can be solved by a computer. | Identifying the requirements of a program.  The halting problem. |
| **Backtracking** | Going back to a previously successful match to find another solution. | Trying an alternative approach if needed. | + Good for solving logic problems and providing artificial intelligence algorithms.  - Only useful for sequential problems. | When one valid move is found in a game, going back to find a better move. |
| **Data Mining** | The analysis of a large amount of data to provide new information. | Looking for patterns within large data sets that are not immediately clear. | + Advantage can be gained if you can spot unexpected trends and patterns.  + Anomaly detection.  - Requires a lot of processing. | Supermarkets found that nappies are often bought by men. Putting beer next to nappies increases the sale of beer. |
| **Heuristics** | Data that provides a suitable but not necessarily accurate solution. A best guess. | Approximating solutions to ensure a balance between time spent on solving the problem and getting to the best possible solution. | + Reduces computation time.  - May not be the best solution. | Estimating congestion when route planning.  A\* algorithm with Manhattan distance to the goal. |
| **Performance modelling** | Carrying out mathematical analysis to assess efficiency. | Knowing how well a solution will perform before full implementation.  Building models to test scenarios. | + Simulations / prototypes predict outcomes.  + Cost effective, time saving & safety-first approach.  - Requires accurate data. | Testing a new exam results system before the day exam results are due to be published. |
| **Pipelining** | The output of one process is the input to another.  Queuing up processes. | Some processes must be achieved one after another. | + Can speed up execution.  - Decisions & branches can mean the pipeline has to be reset, as the next process is no longer the one to be done next. | Mix the cake ingredients, bake the cake, let it cool, add icing.  Fetching the next instruction while decoding the current one. |
| **Visualisation** | Visualisation is a representation of reality using symbols, charts, and colour. | Using diagrams to represent data for analysis.  Modelling scenarios and comparing to visible reality. | + Data is easier to read if it is presented in a visual way.  + May be easier to spot trends, patterns, and relationships between different items of data. | Using diagrams to represent programs: systems diagrams, class diagrams and flowcharts.  Presenting data in unique ways: rainfall in mm expressed as the size of a raindrop icon. |