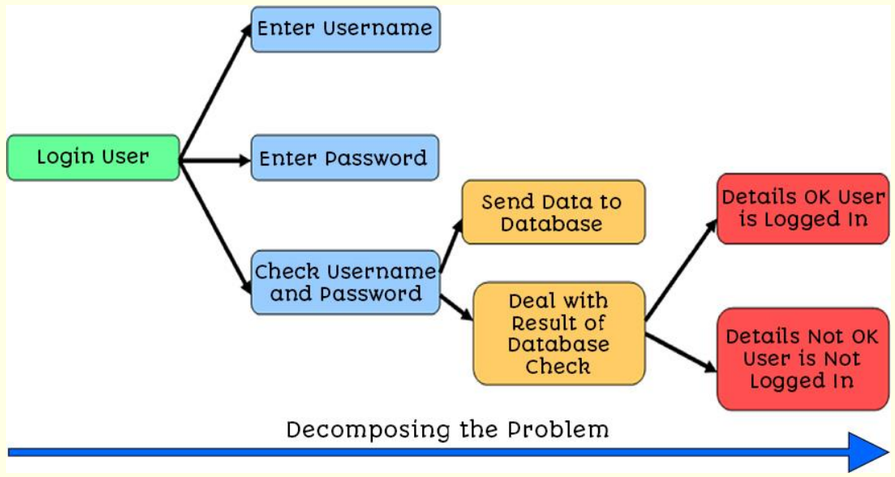
Fold

Abstraction	What is abstraction? Abstraction is the process of removing unnecessary details and focusing only on the essential parts of a system. **Why is Abstraction Important?** • Reduces complexity by hiding unnecessary details. • Saves memory and resources. • Makes systems easier to use, even for non-experts. • Helps in efficient design by focusing on the core elements. • Prevents projects from becoming too large or unmanageable. • High-level programming languages simplify machine code, making it easier to write programs. **Abstraction in Real Life & Computing** • Maps use symbols instead of showing every real-world detail. • Databases simplify real-world objects into structured tables. • The TCP/IP model abstracts complex networking processes. • Object-Oriented Programming (OOP) represents real-world objects as code (e.g., attributes = characteristics, methods = behaviours). **Types of Abstraction** 1. Generalisation – Groups similar elements to find a common solution. 2. Procedural Abstraction – Allows using a function without knowing its internal workings. 3. Data Abstraction – Enables programmers to use complex data structures without worrying about implementation details.	(i) Write a definition of Abstraction A programmer is developing an aeroplane simulator. The user will sit in a cockpit, and the simulated environment will be displayed on screens around them. (ii) Give three potential differences between the abstracted aeroplane simulator and reality.[3] (iii) Identify two reasons why abstraction is used when designing a solution to the problem. [2]	It is a means of hiding detail / only using relevant detail ii) 1 mark for each e.g. • Removal of visual elements such as buildings on the ground • Simplification of controls • Focus on important elements such as weather, height, speed iii) 1 mark for each to max 2 e.g. • Reduce memory requirements • Reduce processing requirements • Simplify the problem being solved

Decomposition	**What is Decomposition?** Decomposition is the process of breaking down a complex problem into smaller, manageable parts. **Why Use Decomposition?** • Smaller parts are easier to solve. • Makes it easier to assign tasks to a team. • Helps in creating subroutines/modules for better organisation. • Allows parallel development, where different teams work on different sections. **Problem Decomposition Process** 1. Break down the problem into smaller parts. 2. Keep dividing until each part is a single task. 3. Each task becomes a subroutine that can be developed and tested separately. 4. Subroutines are combined to create the full solution. 5. Existing modules/libraries may be used to simplify tasks. **Key Technique: Top-Down Design (Stepwise Refinement)** • Problems are divided into multiple levels of complexity. • Development starts with the simplest tasks and builds up to the full solution.  	Write a definition for **Decomposition** Explain how decomposition can aid the design of this program.	• Split the problem down into sub problems • It will creates a more manageable problem / simpler to understand / maintain • can tackle each sub problem independently

Thinking Ahead	**Inputs and Outputs** • Inputs → Data needed to solve a problem (e.g., user input, sensor data). • Outputs → The result after processing the input. • Consider: • How data is entered (keyboard, sensors, etc.). • How outputs are displayed (screen, printer, etc.). • What data structures are needed to store and process inputs/outputs. **Reusable Components** • Code that can be used multiple times in different parts of a program. • Includes functions, subroutines, classes, and libraries. Advantages: • Saves development time. • More reliable (already tested). • Reduces cost and complexity. • Modules can be shared across projects. **Caching** • Stores frequently used data in memory for quick access. • Improves performance and speed by avoiding repetitive data fetching. Examples: • Web caching → Stores visited web pages to load them faster. • Prefetching → Predicts and loads future instructions before needed. Challenges: • Large caches take time to search. • Complex to implement effectively. **Preconditions** • Conditions that must be met before a program runs. • Ensures inputs meet expected criteria. • Can be checked in code or documented separately. Advantages: • Simplifies the program. • Makes subroutines easier to reuse.	**Inputs and outputs** A flight simulator allows a user to take control of a simulated aeroplane. The user can fly the plane in an environment that can simulate different weather conditions and additional planes in the sky. Identify **three** pieces of information that would need to be researched in order to design this simulator. **Reusable components** Explain how programmers make use of reusable components when developing large programs. [2] **Caching** A programmer is developing an aeroplane simulator. The user will sit in a cockpit and the simulated environment will be displayed on screens around them. Describe how caching can be used in the aeroplane simulator. [2]	**Answer:** 1 mark per data item, accept any appropriate, sensible suggestions • Number of other planes that could be in the sky (1) • Speed (1) • Flight path (1) • Altitudes (1) • Rate of acceleration (1) Software is modular (1), an example being an object / function (1). Modules can be transplanted into new software (1) or can be shared at run time (1) through the use of program libraries (1). Answer: 1 mark per bullet e.g. • Store data that has been used in cache/RAM in case needed again • e.g. store design of the weather/a cloud/external environment

Thinking Procedurally	Breaking a problem down. Identifying a number of smaller sub-problems. Determine the order of events. Example: Generating a subject grade requires putting marks into a system, before applying a grade boundary, before printing results.	Mabel is a software engineer. She is writing a computer game for a client. In the game the main character has to avoid their enemies. This becomes more difficult as the levels of the game increase. The computer game allows a user to select a character (e.g. name, gender). They can then choose a level for the game (easy, normal, challenging). The user controls their character by moving it left or right. The character can jump using space bar as an input. If the character touches one of the enemies then it loses a life. The character has to make it to the end of the level without losing all their lives. The game is designed in a modular way. i. One sub-procedure will handle the user input. Describe three other sub-procedures Mabel could create for the given game description.[6]	Answer: i)1 mark per bullet, max 2 per sub-procedure e.g. • Select character (name, gender) • Gives the user options for choosing a character • Choose level • Give the user the choice of level (easy, normal, challenging) and take the user input • Touch enemy • Called to determine if the character touches an enemy • Lose life • Remove a life, if <0 then game over • End level • Move onto next level

Thinking Logically	Decision Making in Algorithms Identifying Decision Points 1. Find where a decision is needed in the program. 2. Determine the conditions that affect the decision. 3. Decide what happens next based on the outcome. Why is this Important? • Helps measure algorithm complexity. • Ensures logical program flow. • Helps in flowchart design (decisions are shown as diamonds).	Eve enjoys playing board games. Her favourite board game is called "Pot Luck". This has a numbered grid of 10 squares by 10 squares. Each square has a number between 1 and 100. Players place their game counters on square 1. A 30-minute timer is set which counts downwards. Each player rolls two 6-sided dice and then moves their game counter that number of squares. Some squares tell the player to pick up a card. These have instructions on, such as 'Move forward 10 spaces'. If the player lands on one of these squares they move according to the instruction on the card. The first player to land on square 100, is announced as the winner. If no winner is announced before the timer runs out, then it is a draw. Logical conditions are checked once a player has rolled the dice. Describe two different logical conditions and how the result will affect the outcome of the game. Logical Condition 1 Condition: Outcome: Logical Condition 2 Condition: Outcome:	**Answer:** 1 mark for condition and 1 mark for outcome to max 4 e.g. • **Condition:** Check if the square has an instruction ... • **Outcome:** ... move the player the number of places specified • **Condition:** Check if they have landed on square 100 ... • **Outcome:** ... Announce the player as the winner • **Condition:** Check if the timer is 0 ... • **Outcome:** ... Announce the game as a draw **Marks:** 4

Thinking Concurrently	Concurrent Processing • Parallel processing is where **multiple processors** are used to complete the same task more quickly. • Concurrent processing is where a **single processor** works on multiple tasks at the same time. • This gives the appearance the tasks are concurrently completed, but in reality they are completed one after the other in quick succession. **Consider what tasks could be processed at the same time.** Advantages of Concurrent Processing • More tasks can be completed in a given time. • Other tasks can be completed whilst awaiting a user decision meaning less time is wasted. Disadvantages of Concurrent Processing • Can take longer to complete a large number of tasks since processes cannot be completed at once. • Some processor time is used to switch between and coordinate processes, reducing overall throughput. • Not all tasks are suited to being completed in this way.	A flight simulator allows a user to take control of a simulated aeroplane. The user can fly the plane in an environment that can simulate different weather conditions and additional planes in the sky. Explain what is meant by 'concurrent processing' and describe one example of how the simulator could make use of it. [4] Concurrent processing Example	Answer: Max 1 for explanation of concurrent programming. Max 3 for each example. Concurrent processing: One process does not have to finish before the other starts (1) Example e.g. Each plane can move independently (1) All move at the same time (1) The weather (1) Wind, rain, direction of air etc. (1) Each element needs to be run simultaneously (1) All need to react to different events (1) Marks: 4 For examples: 1 mark for identifying example. 1 mark for saying how they act concurrently. 1 mark for saying why this is necessary