Thinking Abstractly

The Nature of Abstraction

- Abstraction is the process of separating ideas from reality, hiding unnecessary detail and showing details that are important in context.
- A representation of reality, using symbols to show real life features or irrelevant features left out.
- When designing computer systems, software and interfaces, abstraction often uses methods such as:
  - Symbols
  - Legends
  - Colour Coding
  - Icons
- These are all methods of accentuating (emphasizing) real-life features.
- There are specific subcategories of abstraction, each with its own purpose:
  - Procedural abstraction
  - Functional abstraction
  - Data abstraction
  - Problem abstraction
- Examples of abstract models include flowcharts and top down models.
- Flowcharts are a great example of abstraction of how we can use abstraction when designing a solution to a problem, as a flow chart is simply an abstraction of program code.
  - Many coding concepts such as variables and calculations are abstracted from the programmer.
- Abstraction offers a more simplified view than reality, but accentuates the most important details, for example the nodes and tracks of London's underground.
- There are **two main forms of abstraction**:
  - Representational abstraction - representation arrived at by removing unnecessary detail.
  - Abstraction by generalisation / categorisation - grouping common characteristics to arrive at a hierarchical relationship.

Why Abstraction is Useful

- Simplifies key characteristics.
- Removes extra / unnecessary detail.
- Focuses on the essential aspects of a real-world item.
- Models a real world item, this allows a computational solution.
- Helps decide variables / methods needed.
- Layers of abstraction allow efficient problem solving - focusing on one part of a problem at a time.
- Separate the interface from the implementation.

Layers of Abstraction

- Problem abstraction / reduction
  - Removing details from a problem until it can be represented in a way that is possible to solve.
  - The problem is reduced to one that has already been solved.
- Functional abstraction
  - The result of procedural abstraction is a procedure, not a function.
  - Functions require a further abstraction that also disregards the computational method, this is known as functional abstraction.
- Data abstraction
  - A methodology that isolates how a compound data object is used from the details of its constructions.
  - Involves handling logical data elements which are related to data types.
- Procedural abstraction
  - Abstracting the actual values used in a particular computation as part of a computational pattern or method.
- Network abstraction
  - The OSI model is also an example of abstraction.

Thinking Ahead

- There are four main goals when thinking ahead:
    - Identify the inputs and outputs for a given situation.
    - Determine the preconditions for devising a solution to a problem.
    - Understand the need for reusable program components.
    - Understand the nature, benefits and drawbacks of caching.

When Planning a system:
- Determine the outputs required.
- Determine the inputs necessary to achieve the outputs.
- Consider the resources necessary
- Consider the user's expectations.

Challenges:
- The algorithm must be correct - it must work for all possible inputs.
- The algorithm must be efficient - often involves huge amounts of data, handling millions of records, so finding the most efficient algorithm becomes very important.

Preconditions
- Preconditions are a way to check the right conditions exist to start a program (or sub-program).
- They are like a checklist for a program that should return True if all conditions are met and False if they are not.
- If any precondition is not met then the program should not proceed.
- Advantages of preconditions:
    - With preconditions you read them and you know that if you check the functionality of the program that there may be errors or bugs in the program
    - They are a way for you to check the right conditions exist to start a program (or sub program)
    - They are a checklist for a program that should return true if all conditions are met and false if they are not
    - If any preconditions are not met then the program should not proceed.
    - Making program components reusable
    - Cutting out unnecessary tasks
    - Making programs easier to debug and maintain.

Identifying the Inputs and Outputs
- The advantage of documenting the inputs and outputs is that there is no ambiguity in what must be supplied to the sub-procedure, and what is returned.

Reusable Program Components

Programming Standards
- If program modules are to be reusable, they need to conform to certain programming standards.
- This will help to make them easy to use, for other programmers.
- General standards usually include:
    - Inputs, outputs and preconditions should be documented.
    - Variable identifiers should conform to a standard convention.
    - All variables must be local to the module (except when global variables are required).
    - Documentation should be in a standard format, explaining what the module does, who it was written by and when it was written.
    - Explanations of certain sections should be included where necessary.
    - No module should exclude one page of code.

Libraries
- Libraries are ready-compiled programs, grouped into software libraries, which can be loaded and run when required.
- They can be imported into a user's program.
- Benefits include:
    - They reduce programming time
    - They reduce program code duplication
    - They reduce the amount of testing that is required
    - They help by importing the knowledge of other developers.
    - More standardised approach to code

    - Saves work for the developer
    - Improves interoperability of program components
    - may require fewer developers to work on the project

Reusable Components

- Well-defined and documented functions and procedures may be used in many different programs.
    - They can be added to a library and imported whenever needed.
    - In a large project this will save time and testing, as the functions will have already been written, debugged and tested.

Caching

- Caching is the temporary storage of data and instructions.
- Previously used data is stored in a location that can be quickly accessed to speed up retrieval if needed.

Benefits:

- Quicker searching data items than accessing RAM / secondary storage.
- Quicker access for an item.
- Relies on the item being searched for multiple times.
- Faster accessed to cached resources.
- Shaves costly use of bandwidth.
- Reduces load on web services in client-server environments.

Drawbacks:

- The developer must consider how feasible it is based on the type of information and the number of data items.
- If the query results have been cached, you may see the same available products, but in fact they may have already been sold in the meantime.

Thinking Procedurally

Steps to Solving a Problem

- Many of the programs we use are event driven, however the order in which steps are going to be taken by a user are largely unpredictable.
- That means the modules that make up this software are coded in such a way that they can be accessed by the user in any order.
- On the other hand, some problems involve a very predictable series of steps - with these there is a clear progression in the required steps.
- When attempting to solve a problem, you should consider whether the order of steps required to solve the problem is important or not.
- We can lay out the steps of a program using flowcharts and pseudocode.

Identifying Components of a Problem

- One of the key aspects of thinking procedurally is identifying the individual parts of a bigger problem.
- A common approach to solving a larger problem is to break it down into a top - down modular design (structure diagram) using a method called stepwise refinement.

Top Down Design

- The task the program needs to perform is split into smaller sub-tasks.
- Each of these sub-tasks can be split up into even smaller subtasks if necessary.
- The aim is to end up with subtasks that perform a single function or task and can be tackled as small, largely independent modules.
- The lower level of sub-tasks can be assigned to individual programmers or small teams - they can be written and tested in isolation before being integrated into the overall solution.

Identifying the Components to a Solution

- Each box in a top-down modular design represents a component that needs to be designed in order to produce the overall solutions.
- Each component requires questions such as:
    - What data is required to solve this problem?
    - What on-screen visual elements are needed?
- We can also ask ourselves what outputs will come out of these modules and the preconditions of them.
- With these questions it will be easier to spot other design techniques we can use.
- After decomposing a problem, each component can then be programmed as a single module, procedure, function or method that carries out a single task.

Identifying Sub-Procedures

- Having broken down a problem, we can take the tasks and turn them into a set of functions and procedures.
- Each task will require being coded up by a programmer, who would add the functionality required to make that module work.
- This is also where we would identify any parameters needed for each module - and any returned values in the case of functions.

Advantages of Identifying Sub-Procedures

- Another advantage of identifying all the sub-procedures required to solve a problem is that it becomes easier to spot if a module that performs any of these tasks already exists.
- If there is an existing module that can perform a task, it could be reused and / or customised, saving the developer time and money.

Thinking Logically

Objectives

- Identify the points where a decision has to be taken.
- Determine the logical conditions that affect the outcome of a decision.
- Determine how decisions affect flow through a program.

A Good Algorithm

- Clear and precisely stated steps that produce the correct output for any set of valid inputs.
- Must always terminate at some point.
- Should execute efficiently in as few steps as possible.
- Should be designed in such a way that other people will be able to understand it and modify it if necessary.

Tools for Designing Algorithms

- Hierarchy charts.
- Flowcharts.
- Pseudocode.

Boolean Logic

- **NOT - The output is TRUE if the element is FALSE and vice versa.**
- **AND - The output is TRUE only if both elements are TRUE.**
- **OR - The output is TRUE if either of the elements is TRUE.**
- **XOR - The output is TRUE if either (one) of the elements but not both are TRUE.**

Hand Tracing Algorithms

- A trace table is used to write down the contents of each variable as it changes during execution.
- If the program contains a loop, a helpful technique is to put the loop condition as the first column in the trace table, even if other variables have been defined before it.

Validation Routines

- Checking that a user has entered a value fit for processing can involve tricky Boolean conditions.
- Routines to check that a password is correct also involve several conditions.
  - What happens if the user enters an incorrect input?
  - How many tries are allowed?
  - Will there be some way of giving them a reminder?

Thinking Concurrently

Objectives

- Determine which parts of a program can be tackled at the same time.
- What the benefits and drawbacks are of working concurrently.

Concurrency

- The tendency for things to happen at the same time.
- In Computer Science, it means that multiple computations are happening at the same time.

Parallel Processing

- The terms parallel processing and concurrent processing are sometimes used interchangeably.
- Most commonly, concurrent processing implies that a single processor is switching between separate tasks, so that all tasks appear to be being processed at the same time.
- The distinction is sometimes blurred, for example when many processors are working concurrently on a single task, such as image processing or performing thousands of calculations to calculate the best move in a chess game.

Computational Methods

Problem Recognition

- Identify that there is a problem.
- Identify what the problem is.

Problem Decomposition

- Splitting down a problem into subproblems.
- The aim is to break down the problems until getting to the lowest levels which perform one task.
- Makes tasks easier to translate into code.
- Stepwise refinement can be useful to translate the problem into a top down modular way.
    - Easier to maintain and debug.
    - Easier to write reusable code.
    - Not all programs can be broken down.

Divide and Conquer

- A technique which reduces the size of the problem with every iteration.
- Divide and conquer algorithms split a problem into smaller problems that are easier to solve.
- Individual solutions to these smaller problems must be combined into a larger solution to the original problem.
- This technique is effective but its use is limited, as there are many problems that cannot easily be divided and conquered.

Problem Solving

Backtracking

- Backtracking is the process of incrementally building towards a solution, abandoning partial success when the solution can't be completed and going back to a previously successful match.
- Not suitable for all problems.
- Logic problems, especially those that involve paths or route-finding, can particularly benefit from using backtracking as an approach.

Data Mining

- Data mining is the concept of analysing vast amounts of data gathered from a variety of sources to discover new information and trends.
- It is often associated with the concept of big data.
- Big data refers to extremely large data sets.
- These data sets can be analysed computationally using various data mining techniques to reveal patterns, trends and associations, especially relating to human behaviour and interactions.
- It is not sensible for humans to analyse such vast amounts of data without the assistance of data mining software and tools.
- Turning large quantities of data into useful information / finding patterns within large quantities of information.
    - May include pattern matching algorithms.
    - May involve anomaly detection algorithms.
- Data mining is used by companies to help them maximise their profits.
- It's also used to plan for future eventualities.
- Applications of data mining include:
    - Weather modelling
    - Business and Economics
    - Stock Market
    - Science and Engineering
    - Medical Research
    - Law Enforcement

Big Data

- Implies huge amounts of data are collected and stored.
- Defined by three major features, known as the 3V's:
    - Volume
    - Variety
    - Velocity
- Parallel computing in which algorithmic tasks are executed concurrently on a cluster of machines or supercomputers, is fundamental to managing big data tasks.

Heuristics

- Heuristics is an approach to solving problems that allows or even encourages us to make use of our experience and find a solution that can be considered "good enough".
- Sometimes, trying to find an exact answer to a problem may be unrealistic or even totally impractical due to hardware limitations.
- In these situations, a solution that has a high probability of being correct may be acceptable.
- Designed for solving a problem quicker when classic methods are too slow, or finding an approximate solution when classic methods fail to find an exact solution.
- By their very nature, heuristic approaches and solutions do not guarantee the best solution, but they will generally produce results that are good enough.
- May not be 100% reliable.
- That is why heuristic methods are often referred to as rule of thumb.
- Using heuristic methods, many intractable problems - which would normally be impossible to solve due to a variety of limitations, can be solved by sacrificing the optimal answer for the one that is good enough.
- Applications include:
    - Routing messages across the internet
    - Building circuit boards
    - Transportation
    - Virus checking
    - DNA analysis
    - Artificial intelligence

Performance Modelling

- Performance modelling is the process of approximating how well models perform using mathematics.
- This method of solving problems is based around the use of simulations and mathematical approximations within those simulations without having to perform detailed testing, which may otherwise be prohibitive in terms of time and cost.
- One measure of efficiency is the big-o notation, which measures the suitability of algorithms in terms of execution time and space.
- This strategy is often used by developers of online, massively multiplayer games.
- A popular online game may have hundreds of thousands of players trying to log in and use the system once it has gone live.
- Trying to test how the servers will handle this sort of traffic and load up before release is tricky.
- Instead, once a game is nearing release, developers often open up the game as part of a f2p beta testing phase.
- The developer can limit the number of people connecting to the servers, monitor the situation and then use performance modelling to calculate whether the servers can cope under the greater strain going forward.

Pipelining

- Pipelining means splitting a large task into manageable chunks and overlapping these smaller processes to speed up the overall process.
- For example, executing a program in the CPU - one processor core could be fetching an instruction while other cores are decoding and executing other instructions.
- Pipelining is an implementation technique where multiple instructions are overlapped in execution.
- Instructions enter the pipeline at one end, and at each stage part of the instruction is completed and moves to the next stage, while another instruction enters the pipeline, rather like an assembly line.

Visualisation

- The way we visualise a problem can have a major impact on the journey to a workable solution.
- Visualisations allow us to create a mental image of what a program will do or how it will work.
- As humans, we naturally respond well to visual representations of complex situations.
- In a block of text, we can describe a landscape in minute detail. We can then visualise the scene being described - this is what happens when we read books.
- We often use flow diagrams to illustrate the logic of an algorithm before turning it into pseudocode.