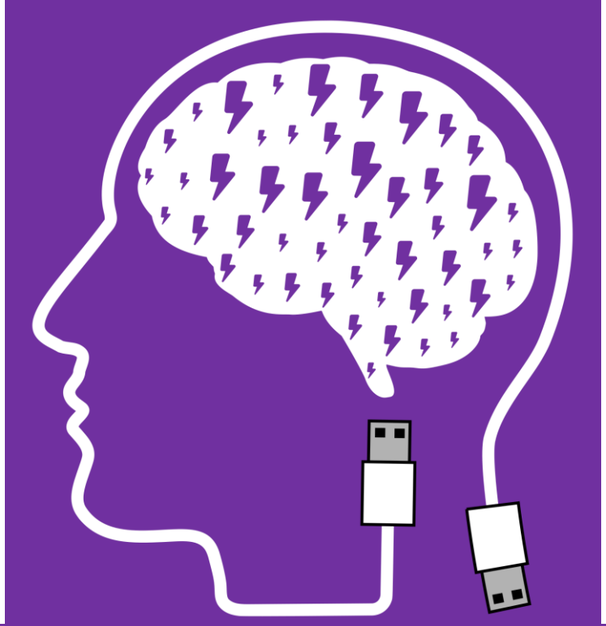
Unit 2.1 Elements of Computational Thinking

Preconditions

- Things which are needed before the program can run.
- The code expects the information passed to it to meet certain criteria.
- The code may test for these when it is run.
- They may instead be included within documentation.
- Including this information within documentation reduces the complexity of the program and makes it easier to use.
- Preconditions make it easier to reuse subroutines.



Abstraction

- Removing unnecessary detail.
- Representing the key elements of the problem.
- Must consider what information is actually relevant to the problem at hand.
- Complex problems can be split into several layers of abstraction.
- Higher layers are closer to the user, possibly providing a user interface.
- Lower levels interact with the computer.

Abstraction and Reality

- Abstraction is more simplified than reality.
- Real world items are simplified into computer structures such as a table, variable or database.
- Objects used in object oriented programming can be an abstraction of real world entities.
- Attributes can represent the characteristics of a real world object.
- Methods can represent the actions a real world object may perform.

Creating an Abstraction Model

- What problem needs to be solved?
- Who will use the model?
- How will the model be used?
- Which are the key elements of the problem for the people using the model and how they will use it?

Abstraction by Generalisation

- Similar elements of a problem may be grouped together.
- This allows common problems to be categorised.
- They can then be solved with a common solution.

Decisions Affecting Program Flow

- There may be many routes through a program.
- Decisions by the user will affect the route taken.
- It is important to identify places where the user will need to make a decision and plan for the decisions they may make.

Procedural Abstraction

- Allows a programmer to use a function without understanding the detail of its implementation.
- Used with data structures and in decomposition.
- Models the purpose of a subroutine without considering how it does what it does.

Inputs and Outputs

- An input is any data required to solve the problem.
- These may be entered by the user, or obtained from hardware such as a sensor.
- Outputs are the solutions to the problem which are returned.
- They can only be produced once the input has been processed.
- It is important to consider the methods used to capture data from the user and to present it back to them.
- Think about the data structures used.
- Think about the devices used.
- Think about what outputs are needed first.
- Use this information to consider what inputs are needed to produce the required output.

Decision Making

- There are many decisions involved with making and designing programs.
- It is important to consider these decisions carefully.
- Often, the available choices for a decision may be limited, simplifying the decision.
- Identifying the decisions which need to be made allows information to be gathered on potential choices.
- In flow charts, decisions are represented by diamonds.

Caching (ALEVEL ONLY)

- Values or information can be stored in memory after use.
- This makes it quicker to retrieve them if they are needed again.
- Web pages are also cached in this way to improve load times and reduce bandwidth usage.
- Prefetching uses an algorithm to predict which instructions may be needed next and store them in cache before they are needed.
- This reduces the need to wait for an instruction to be loaded.
- The accuracy of the algorithm's predictions influences the effectiveness of this technique.
- A large cache can take a long time to search.
- Caching and prefetching can be difficult to implement.

Data Abstraction

- Programmers may use complex data structures without needing to understand how they are implemented in detail.
- How data is being stored and filtered.

Problem Decomposition

- Breaking down a large problem into smaller parts.
- These smaller parts are easier to solve.
- The smaller parts are easy to divide among a team.
- Top down design, also called stepwise refinement is often used to do this.
- This technique divides a problem into levels of complexity.
- Problems are broken down over and over until each problem is a single task.
- Each task can then be solved with a single subroutine.
- Subroutines can be tested and developed separately.
- Consider how each subroutine is implemented.
- The subroutines need to be joined to form the whole solution.
- Start with the lowest level components and work up.
- Some tasks may be solved with an existing module or library.

Working Concurrently

Concurrent Thinking

- Considering more than one task at the same time.
- All the tasks need not be actively worked on at the same time.
- Giving parts of your time to different tasks throughout the day.
- Parts of multiple problems are often related, allowing them to be solved 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.

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.

Reusable Program Components

- Common functions can be packaged into a library.
- This makes it easier to reuse them throughout a project.
- Abstract data structures, subroutines and classes can all be reused in this way.
- Decomposition is used to indicate where components of an existing program can be reused.
- Reusable components have already been tested and so are more reliable.
- They make development less time consuming and therefore less costly.

The Need for Abstraction

- Allows those who are not experts in a field to use systems by hiding more complex information which is irrelevant to using the system.
- Allows more efficient design by encouraging focus on the core elements of a problem.
- Reduces the time spent on a project.
- Prevents a project becoming too large or complex.
- Low-level programming languages directly interact with hardware but are hard to write so high-level languages abstract the machine code that is executed when a program is run.
- The TCP/IP model is an example of abstraction in networking.

The Order of Steps

- It is important to consider the order in which operations are performed.
- Certain inputs may be required before processing.
- Inputs may need to be validated, this must occur after the input is received and before it is processed.
- It may be possible for several subroutines to be executed at the same time.
- Also consider how subroutines interact with one another.
- Code should be written to prevent operations occurring in an order which would cause an error or prevent the program from functioning as intended.

Conditions Affecting a Decision

- Effectiveness
- Convenience
- Cost
- Efficiency
- Relevance
- Available skills and resources
- All these conditions are important.
- Some may be more important to a particular decision.