Definitions and Concepts for OCR Computer Science GCSE

Topic 2: Computational thinking, algorithms and programming

2.1 – Algorithms

Decomposition: The process of breaking down a problem into smaller, more manageable sub-problems that each accomplish a specific task.

Abstraction: The process of reducing complexity by hiding unnecessary details to focus on the essential parts of a problem, system, or solution.

Algorithmic Thinking: The ability to create a clear and logical set of instructions (an algorithm) that can be followed to find the solution to a problem.

Pattern Recognition: The process of identifying similarities or recurring features in problems so that known solutions and techniques can be applied to solve them.

Inputs: The data an algorithm or program receives.

Processing: How the data is transformed or calculated during a program.

Outputs: The result or outcome produced by the algorithm, often presented to the user.

Pseudo-code: A simplified, informal way of describing an algorithm that is closer to human language than programming code, but structured like code.

Program Code: A set of instructions written in a high level programming languages (e.g., Python, C#) that tells a computer how to perform a specific task or solve a problem.

Flowcharts: Diagrams that use symbols / shapes to represent the steps and decision-making processes of an algorithm.

Syntax Error: An error which breaks the grammatical rules of the language.

Logic Error: An error in the code which causes it to produce unexpected or incorrect results, despite the code still running.

Trace Table: A table used to record the values of variables as a program is executed, step by step, to help follow the logic of an algorithm and check for errors.

Algorithm: A step-by-step set of instructions that can be followed to solve a problem.

Searching Algorithm: An algorithm used to find a specific item within a list.

Linear Search: A searching algorithm which searches for an item by checking each item in the list one by one, beginning at index 0.

Binary Search: A searching algorithm which searches for an item by repeatedly dividing sorted lists at the midpoint.

Sorting Algorithm: An algorithm which is used to arrange data in a specific order, such as ascending or descending.

Bubble Sort: A sorting algorithm which repeatedly goes through a list and compares 2 adjacent items, swapping them if they are in the wrong order.

Merge Sort: A sorting algorithm which continuously splits a list in half until it consists of several sublists of length 1, then merges these sublists back together to form a sorted list.

Insertion Sort: A sorting algorithm which inserts an unsorted list of items into their correct position in a list.

2.2 – Programming fundamentals

Variable Declaration: Creates a variable, a named location in memory, to store data.

Constant Declaration: Creates a value that does not change while the program runs, such as Pi. These are typically given an all-caps identifier.

Assignment: Setting or updating a value in a variable.

Input: Receiving data from the user.

Output: Displaying data or information to the user.

Sequence: Instructions are executed in the order they are written.

Selection: Decisions, such as IF-ELSE statements that cause branching.

Iteration: The repetition of a block of code, such as FOR and WHILE loops.

Count-Controlled Iteration: A type of iteration where a block of code is repeated a finite, predetermined number of times, e.g., FOR loops.

Condition-Controlled Iteration: A type of iteration where a block of code is repeated until a condition is true or false. The number of repetitions is not fixed in advance, e.g., WHILE.

Arithmetic Operations: Basic mathematical calculations that can be performed in a programming language, such as + (plus).

Comparison Operations: Used to compare different items of data, and either return True or False, such as == (equal to).

Boolean Operations: Logical operations that compare or combine Boolean values and are used in conditions to control the flow of a program, consisting of AND, OR, and NOT.

Data Type: Defines the kind of data a variable can hold, as well as how a program will store, process, and display that data, and the operations which can be performed on it.

Integer: Whole numbers.

Float: Numbers which may include a decimal part.

Boolean: Used in decision making processes, with the only two values being True/False.

Character: A single symbol or letter.

String: A sequence of characters.

Casting: Refers to changing the data type of a variable.

String Manipulation: Performing operations on strings, including measuring their lengths, concatenating strings, or slicing strings.

Concatenation: Combining two or more strings to form a longer one.

Slicing: The process of extracting a section (substring) of a string by specifying a start and end position (often as an index).

File Handling: Reading or writing to an external file.

Data Structure: A way of organising and storing data within a structure to be used efficiently.

Record: A type of data structure which groups different types of data together.

Array: A fixed-length data structure which can only store items of the same data type.

Database: A database is used to store large amounts of data into organised tables.

SQL: Structured Query Language (SQL) is used to search for, manage, and manipulate data in a database.

Subprogram: A section of code which performs a specific task, and must be called whenever it is needed to carry out that task.

Function: A type of subprogram which performs a specific task and returns a value.

Procedure: A type of subprogram which performs a specific task and does not return a value.

Local Variable: A variable defined inside of a subprogram, that can only be used within that same subprogram and is overwritten in memory when the subprogram ends

Global Variable: A variable defined in the main program, that can be used anywhere in the program.

Random Number Generation: The ability to produce unpredictable numeric values within a specified range.

2.3 – Producing robust programs

Defensive Design: Designing and creating programs so that they are able to handle unexpected or erroneous inputs by anticipating the program's misuse.

Authentication: The process of determining the identity of a user.

Input Validation: The process of checking that any input is appropriate for its use.

Presence Check: Checks if data has actually been entered and not left blank.

Range Check: Checks that data falls within a specified range.

Length Check: Checks if a specified number of characters have been entered (often over or under a limit).

Naming Convention: Appropriately naming all variables and subroutines to reflect their purpose and improve code readability.

Iterative Testing: Testing which takes place throughout development, the results of which are used to improve the program.

Final Testing: Testing performed at the end of development to ensure the program meets all requirements, works correctly under all expected conditions, and is ready to release.

Syntax Error: An error which breaks the grammatical rules of a programming language, preventing the program from running.

Logic Error: An error in the program's design or logic, causing it to produce an unexpected or incorrect output, whilst still running.

Normal Test Data: A typical, valid input.

Boundary Test Data: An input on the edge of a range.

Erroneous Test Data: An input outside a range or of the incorrect data type.

2.4 – Boolean logic

Logic Gate: An electronic component that performs a basic Boolean operation on one or two input signals to produce a single output signal.

Logic Circuit: A combination of interconnected logic gates designed to perform a specific Boolean function.

Boolean Logic: A branch of algebra, where values take either TRUE (1) or FALSE (0), used in decision making.

Truth Table: Used to show the different outputs of Boolean expressions for all possible input combinations.

AND Gate: Returns TRUE if both inputs are TRUE.

OR Gate: Returns TRUE if either input is (or both are) TRUE.

NOT Gate: Reverses the input.

2.5 – Programming languages and Integrated Development Environments

Programming Language: Used to write instructions which computers can execute.

High-Level Language: A programming language closely resembling human language, making it possible for humans to read and write code using structured English.

Low-Level Language: A language closer to machine code, often directly understood by computers but harder for humans to understand.

Translators: Used to convert high level languages to low level languages so that programs can be executed.

Compilers: Translates entire high level programs at once, producing a platform-specific executable file.

Interpreters: Translates high level programs line-by-line, checking for errors as they go.

IDEs: Integrated Development Environments (IDEs) are platforms used by programmers which contain features to make creating clear and maintainable code easier.

Editors: The environment in an IDE which allows the user to write and develop code.

Error Diagnostics: A feature of an IDE which identifies errors, and provides information on where the error is in the code, what is wrong, and possible solutions to fix the error.

Run-time Environments: A platform which allows programs to run, using a virtual computer to simulate the program running on multiple different environments.