Topic 4—Computational thinking, problem-solving and programming (45 hours) ***Notes by Dave Mulkey, 2015, Germany***

4.1 General principles (10 hours)

This should not be taught as a separate topic but must be incorporated and connected to all sections—especially flow charts, pseudocode and programming in the SL/HL core and abstract data structures (HL extension). It is essential that these elements are not addressed in isolation—they have to be approached as a whole.

The basic ideas and their application should be illustrated with non-computer examples. Each basic idea should then be practised in specific algorithmic contexts using concepts drawn from flow charts, pseudocode and programming. The teacher support material illustrates examples such as the home/locker/knapsack for thinking ahead.

	Assessment statement	**Ob**	**Teacher's notes**	**Explanations and Examples**
Thinking procedurally				
4.1.1	Identify the procedure appropriate to solving a problem.	2	This includes identifying the steps and putting them in the correct order. Such as recipes, block-arrow-block-arrow. **LINK** Connecting computational thinking and program design, introduction to programming.	This means to write a set of steps for an ALGORITHM that solves a problem reliably. These could be written in **PSEUDOCODE** or in a FLOW-CHART. For example: Add up all numbers from 1 to 99 SUM = 0 loop NUM from 1 to 99 SUM = SUM + NUM end loop output SUM
4.1.2	Evaluate whether the order in which activities are undertaken will result in the required outcome.	3	Links to problems presented to the student in other areas of the syllabus. **LINK** Thinking ahead, thinking concurrently. Connecting computational thinking and program design, introduction to programming. **MYP** Technology, step-by-step instructions.	This means TESTING an algorithm using SAMPLE DATA. For example, the following should add up all positive numbers in an array containing 5 numbers. Test with NUMS = [5 , -5 , 0, -2 , 2] loop C from 0 to 4 SUM = 0 SUM = SUM + NUMS[C] end loop

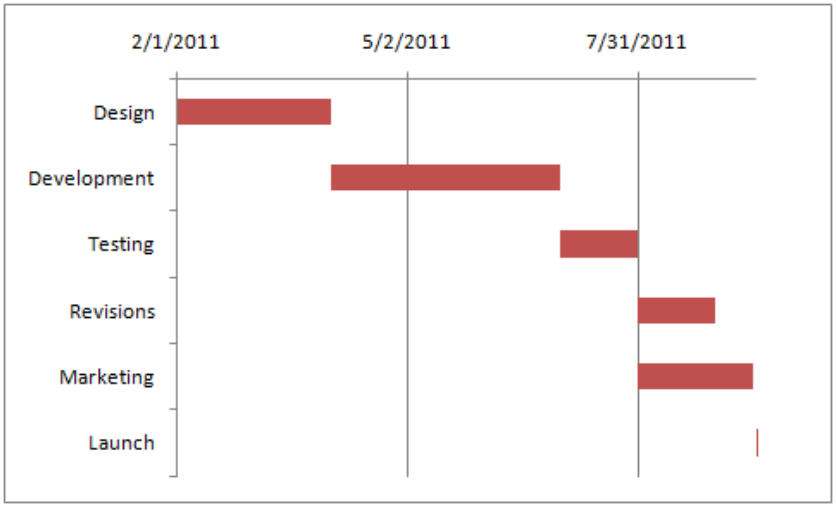
				---> answer : result is SUM = 2 but it should produce SUM = 7 The SUM = 0 command should be before the loop command
4.1.3	Explain the role of sub-procedures in solving a problem.	3	Constructing procedures that can then be referred to by their identifier. **LINK** Abstraction, connecting computational thinking and program design, introduction to programming.	If the programming language provides pre-written procedures, then the algorithm can execute a pre-written command instead of the programmer writing lots of code.s For example, if the language contains a SUM command for adding up an array, then it's not necessary to write a loop. Instead, just write : ANSWER = SUM(NUMBERS)
Thinking logically				
4.1.4	Identify when decision-making is required in a specified situation.	2	Links to procedural thinking—alternative procedures. **TOK** Reasoning as a form of decision-making. **LINK** Connecting computational thinking and program design, introduction to programming.	In PSEUDOCODE, decisions are written as IF...THEN… In FLOW-CHARTS, use a Diamond to represent a decision.
4.1.5	Identify the decisions required for the solution to a specified problem.	2	Different actions are taken based on conditions. **LINK** Connecting computational thinking and program design, introduction to programming. **AIM 4** Applying thinking skills to identify and resolve a specified complex problem.	Common decisions are: Strings : if ANSWER = "YES" then… Numbers : if AGE > 21 then … Boolean : if FOUND = TRUE then ...
4.1.6	Identify the condition associated with a given decision in a specified problem.	2	Testing conditions, iteration. Identifying and constructing the conditions—AND, OR, NOT relationships—Boolean tests.	Compound BOOLEAN conditions us AND, OR, NOT For example : if AGE >= 13 AND AGE <=18 then SCHOOL = "High School"

			LINK Connecting computational thinking and program design, introduction to programming.	end if
				
				if TODAY = "SAT" OR TODAY = "SUN" then
				output "Weekend"
				end if
				
				if NOT(PASSWORD = "magic") then
				output "Wrong Password"
				end if

4.1.7	Explain the relationship between the decisions and conditions of a system.	3	IF … THEN … ELSE
			LINK Connecting computational thinking and program design, introduction to programming.
				For example:
				if PASSWORD = "magic" then
				CORRECT = TRUE
				else if PASSWORD = "MAGIC" then
				CORRECT = TRUE
				else
				CORRECT = FALSE
				end if

4.1.8	Deduce logical rules for real-world situations.	3	**LINK** Connecting computational thinking and program design, introduction to programming.	For example, in a black-jack game:
				
				if MYTOTAL = 21 AND MYCARDCOUNT = 2 then
				output "I win"
				else if MYTOTAL > 21 then
				output "I Lose"
				else if DEALERTOTAL > 21 then
				output "I Win"
				else if MYTOTAL > DEALERTOTAL then
				output "I Win"
				else
				output "I Lose"
				end if

Thinking ahead				

4.1.9	Identify the inputs and outputs required in a solution.	2	This is connected to TESTING - a TEST-CASE must specify the expected INPUT and the required OUTPUT	For example: Algorithm should ADD UP all positive numbers in an array Test Case A Input = [5 , -2 , 0 , -5 , 3] Output = SUM = 8 Test Case B Input = [-1 , -2 , -3] Output = 0
4.1.10	Identify pre-planning in a suggested problem and solution.	2	Gantt charts. Pre-ordering. Pre-heating an oven. Home/locker/knapsack. Caching/pre-fetching. Building libraries of pre-formed elements for future use. **LINK** Thinking procedurally, thinking concurrently. Connecting computational thinking and program design, introduction to programming.	Gantt Chart - for project planning 
4.1.11	Explain the need for pre-conditions when executing an algorithm.	3		Examples : (a) Before comparing two Strings containing dates, the dates probably need to be in a specific format, like "dd-mm-yyyy" (b) BINARY SEARCH can only be performed in an array if the array is already SORTED - so a precondition of is having a SORTED array

				(c) Before converting a String containing a Binary number to decimal, the String must contain at least one character, and all characters must be 0 or 1.
4.1.12	Outline the pre- and post-conditions to a specified problem.	2	For example, cooking a dish for a meal. All ingredients available before starting to cook. A place to eat the food.	
4.1.13	Identify exceptions that need to be considered in a specified problem solution.	2	For example, identify the pre-conditions for calculating the end-of-year bonus when not all employees have worked for the company for the whole year. **LINK** Connecting computational thinking and program design, introduction to programming.	For example: (a) Solving Ax^2 + Bx + C = 0, there is NO SOLUTION if B^2-4AC < 0 - that's an exception (b) When searching in a Linked-List, it could happen that the HEAD pointer is NULL. Then the search must end immediately, returning NOT FOUND
Thinking concurrently				
4.1.14	Identify the parts of a solution that could be implemented concurrently.	2	Could include computer systems or real-life situations. **LINK** Thinking ahead, thinking procedurally. Connecting computational thinking and program design, introduction to programming.	For example : We cannot SEARCH an array at the same time that a different algorithm is SORTING the array But: We can ADD UP all the numbers in an array at the same time that a different algorithm is searching for the largest and smallest values, and a third algorithm is copying all the values from the array into a TextArea on the computer screen.
4.1.15	Describe how concurrent processing can be used to solve a problem.	2	For example, building a house, production lines, division of labour. Students will not be expected to	Algorithms are not required, but a Gantt chart would be appropriate. Concurrent processes are are the ones with bars that overlap.

			construct a flow chart or pseudocode related to concurrent processing.	
4.1.16	Evaluate the decision to use concurrent processing in solving a problem.	3	**LINK** Thinking ahead, thinking procedurally. Connecting computational thinking and program design, introduction to programming.	Concurrent processing makes sense when it is possible to use multiple devices simultaneously, hence reducing the amount of time required. Usually, WRITE commands cannot be done concurrently with anything else, whereas READ commands can be done concurrently.
Thinking abstractly				
4.1.17	Identify examples of abstraction.	2	Selecting the pieces of information that are relevant to solving the problem. **LINK** Thinking ahead.	Abstraction means simplifying reality so that it can be represented (stored) inside a computer. For example, a road map can be represented with codes for the starting and ending points of the road, plus a number that tells the length of the road. It is not necessary to actually have a picture of the turns in the road. A famous problem is the Bridges of Koenigsberg, where only the starting and ending points are saved.
4.1.18	Explain why abstraction is required in the derivation of computational solutions for a specified situation.	3	Students should be aware of the concept of objects, for example, the use of collections as objects in the design of algorithms. **LINK** ● Databases: tables, queries ● Modelling and simulation: an abstraction of reality ● OOP: classes, sub-classes ● Web science: distributed applications	In addition to encoding data items, for example as numbers, DATA-STRUCTURES provide a chance to represent RELATIONSHIPS between data items. For example, an ARRAY or a LINKED-LIST can record data items in a specific order. A TREE can save a HIERARCHICAL STRUCTURE - for example representing a mathematical formula or a directory tree. But neither a LIST nor a TREE can correctly represent the WEB, which has arbitrary numbers of links from any node, as well as links going in both directions between nodes.

4.1.19	Construct an abstraction from a specified situation.	3	There is no need to use code. Levels of abstraction through successive decomposition. A school can be decomposed into faculties. A faculty can be decomposed into departments. **LINK** Thinking ahead, thinking procedurally. Connecting computational thinking and program design, introduction to programming.	This might be done using a DIAGRAM, like a tree with lines linking nodes together. Or a 2-dimensional array (table).
4.1.20	Distinguish between a real-world entity and its abstraction.	2	**TOK** The map as an abstraction of the territory.	For example, an ID number for a student and the student's NAME are not the same as the actual student. Usually, abstractions only represent part of the important data. A student also has a birthdate, a phone number, height, weight, nationality, etc. Even with all that data, we still would not have a PICTURE of the students face, and we would not know what they ate during the past week. The ingredients shown on a cereal box might represent all the significant nutritional information, but they do not tell us how the cereal will taste, or whether it is crunchy.

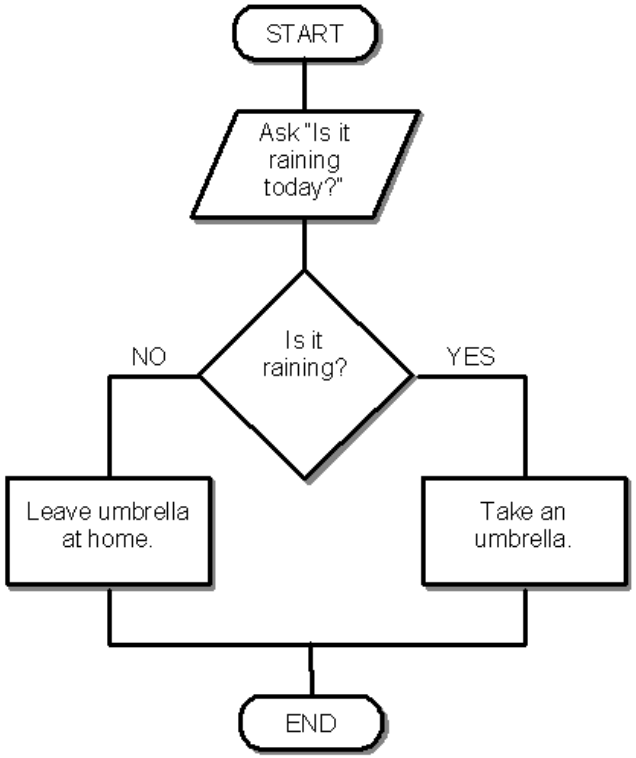
4.2 Connecting computational thinking and program design (22 hours)

The focus of this topic is how an understanding of programming languages enhances the students' understanding of computational thinking and provides them with opportunities for practical, hands-on experience of applying computational thinking to practical outcomes.
In externally assessed components questions will be presented using flow charts and/or pseudocode as outlined in the approved notation sheet. Answers will only be required in pseudocode.
Students must be given the opportunity to convert algorithms into working code that is executed and tested.
Working code will not be assessed in the externally assessed components.

	Assessment statement	Ob	Teacher's notes	Explanations and Examples

4.2.1	Describe the characteristics of standard algorithms on linear arrays.	2	These are: sequential search, binary search, bubble sort, selection sort.	Sequential Search
				Start at the beginning
				Check each position in the array
				If target item is found then
				output POSITION where item was found
				else
				move to next item
				
				Binary Search
				Start in the middle of the array
				if target item is found then
				output POSITION where item was found
				else if target < current item then
				start again in the middle of the first half of list
				else
				start again in the middle of the 2nd half of list
				
				Bubble Sort
				N = number of items in the array
				Search through entire list N times
				At each position, compare DATA[X] to DATA[X+1]
				swap items X and X+1 if they are not in order
				continue to end of list
				Next pass
				
				Selection Sort
				find BEST item in list
				swap it to beginning of list
				find BEST item starting in second position

				swap into 2nd position repeat at 3rd position, then 4th position, etc Requires N passes, but each pass is 1 item shorter
4.2.2	Outline the standard operations of collections.	2	These are: addition and retrieval of data.	A collection is an UNORDERED list of data - a "SET" The collection is a data-structure that has these operations: .addItem(data) .resetNext() = start at the beginning .hasNext() → tells whether there is another item in the list .getNext() → retrieves a data item from the collection .isEmpty() → check whether collection is empty
4.2.3	Discuss an algorithm to solve a specific problem.	3	Students should be expected to discuss the differences between algorithms, including both standard and novel algorithms. For example, discussing the advantages and disadvantages of using a binary search as opposed to a sequential search.	A binary search is faster - O(log N), but can only be performed in a SORTED list A sequential search is slower - O(N) but can be performed whether the list is sorted or not A bubble sort can "quit early" if no swaps are made in a pass. But it makes lots of swaps. A selection sort must always perform N passes - it cannot "quit early". But it makes fewer swaps - maximum of N swaps Both Bubble and Selection sorts are O(n^2)
4.2.4	Analyse an algorithm presented as a flow chart.	3	Examination questions may involve variables, calculations, simple and nested loops, simple conditionals and multiple or nested conditionals. This would include tracing an algorithm as well as assessing its correctness.	Be sure to learn the correct shapes of boxes: Input/Output = parallelogram Process(calculation) = rectangle Decision (if..then) = diamond

			Students will not be expected to construct a flow chart to represent an algorithm in an externally assessed component. **MYP** Mathematics: using flow charts to solve problems in real-life contexts, patterns and sequences, logic, algorithms. **MYP** Technology: design cycle (inputs, processes, outputs, feedback, iteration).	Start/end/connect = circle 
4.2.5	Analyse an algorithm presented as pseudocode.	3	Examination questions may involve variables, calculations, simple and nested loops, simple conditionals and multiple or nested conditionals. This would include tracing an algorithm as well as assessing its correctness. **MYP** Mathematics: using flow charts to solve problems in real-life	Practice here: **Pseudocode Practice Tool** Here are some old IB questions presented with Pseudocode. Old Exam Questions These questions are REALLY OLD and require some updating still.

			contexts, patterns and sequences, logic, algorithms. **MYP** Technology: design cycle (inputs, processes, outputs, feedback, iteration).	
4.2.6	Construct pseudocode to represent an algorithm.	3	**MYP** Mathematics: using flow charts to solve problems in real-life contexts, patterns and sequences, logic, algorithms. **MYP** Technology: design cycle (inputs, processes, outputs, feedback, iteration). **AIM 4** Demonstrate thinking skills to represent a possible solution to a specified complex problem.	Practice here: **Pseudocode Practice Tool**
4.2.7	Suggest suitable algorithms to solve a specific problem.	3	Suitable algorithms may include both standard algorithms and novel algorithms. Suitable may include considerations of efficiency, correctness, reliability and flexibility. Students are expected to suggest algorithms that will actually solve the problem successfully. **LINK** General principles of computational thinking, introduction to programming.	For example: If PRICES and NAMES and INVENTORY are parallel arrays, write an algorithm that finds all the items where INVENTORY is below 10 items, and adds 20% to the PRICES of those items.
4.2.8	Deduce the efficiency of an algorithm in the context of its use.	3	Students should understand and explain the difference in efficiency between a single loop, nested loops, a loop that ends when a condition is met or questions of	Big O notation is not required, but makes things simpler: `Sequential Search → O(n) (n is the length of the list)`

			similar complexity. Students should also be able to suggest changes in an algorithm that would improve efficiency, for example, using a flag to stop a search immediately when an item is found, rather than continuing the search through the entire list.	Binary Search → O(log n) (that's log base 2) Bubble Sort → O(n^2) [HL] Travelling Salesman Problem → O(n !) (n=number of cities)
4.2.9	Determine the number of times a step in an algorithm will be performed for given input data.	3	Examination questions will involve specific algorithms (in pseudocode/flow charts), and students may be expected to give an actual number (or range of numbers) of iterations that a step will execute.	"number of steps" is called ITERATIONS In nested loops, multiply the lengths of each loop to determine iterations of inner-most command

4.3 introduction to programming (13 hours)

	Assessment statement	Ob	Teacher's notes	Explanations and Examples
Nature of programming languages				
4.3.1	State the fundamental operations of a computer.	1	These include: add, compare, retrieve and store data. Complex capabilities are composed of very large numbers of very simple operations.	ADD means numerical addition This is referring to Machine Code - the language that a CPU actually understands = NATIVE code
4.3.2	Distinguish between fundamental and compound operations of a computer.	2	For example, "find the largest" is a compound operation.	Add accumulator plus 1 → fundamental Store accumulator into memory → fundamental Storing 1,3,5,7,...,99 → compound (uses a loop) Comparing two Strings → compound, as it must loop through the Strings and compare many individual characters
4.3.3	Explain the essential features of a computer language.	3	For example, fixed vocabulary, unambiguous meaning, consistent grammar and syntax. **TOK** Language and meaning.	English is not a computer language because meaning is NOT unambiguous - for example "store" has several different meanings Example computer languages: Java, Javascript, Python, Fortran, Basic, C++
4.3.4	Explain the need for higher level languages.	3	For example, as the human needs for computer systems have expanded it is necessary to abstract from the basic operations of the computer. It would take far	High Level code is easier to write, as one single command like **output ARRAY** might print out 100 numbers, but low level languages require a loop for this task. High Level code may provide more sophisticated concepts,

			too long to write the type of systems needed today in machine code.	like Object Oriented Programming, that make programming easier and quicker and more reliable High Level language tools often provide sophisticated and automated error-checking, -prevention, and -handling High Level code also supplies LIBRARIES, such as an Email CLASS that can send email messages
4.3.5	Outline the need for a translation process from a higher level language to machine executable code.	2	For example, compiler, interpreter, virtual machine.	CPUs and Compilers and Virtual Machines Compiler - reads the entire program (source code) searching for syntax errors. If no errors are found, it translates the High Level source code into low level machine code, that can run directly on the CPU. Interpreter - reads one command at a time and then executes it immediately. So if there is an error later in the program, the program will start but fail in the middle. Interpreters generally provide less debugging help than compilers. A Virtual Machine is a software level that allows the compiler or interpreter to produce output code that does not need to run directly on the CPU, but rather in the Virtual Machine. Java provides a virtual machine that is an identical run-time environment on all platforms, so Java does not need to be converted to native code. By creating a virtual machine for each platform, Java is made "cross-platform" by design. Unfortunately, this does not work as perfectly as we might expect, due to minor hardware and OS differences.

Use of programming languages Sub-programmes and objects support abstraction, which facilitates: ease of debugging and maintenance, reuse of code, modularity. There is no programming language specified in the SL/HL core. However, students must use a language that supports the basic constructs on the approved notation sheet .				
4.3.6	Define the terms: variable, constant, operator, object.	1		variable = a name that represents a value constant = a value that cannot change during run-time operator = numerical operations, String operations, logical (boolean) operations e.g. operations on primitive data types object = a collection of data and methods, created from a design (class), allowing multiple INSTANCES An object has a REFERENCE VARIABLE that "points to" the contents of the object
4.3.7	Define the operators =, ≠, <, <=, >, >=, mod, div.	1	**LINK** Approved notation sheet.	div = integer division, no fractional part in the result mod = the remainder from an integer division The others are COMPARISON operators - they are also BOOLEAN operators as they produce TRUE or FALSE results.
4.3.8	Analyse the use of variables, constants and operators in algorithms.	3	For example, identify and justify the use of a constant as opposed to a variable in a given situation. **MYP** Mathematics: forms of numbers, algebra—patterns and sequences, logic, algorithms.	Local variable = created inside a method, with SCOPE that extends only to that method - it is not available outside that method Global variable = created outside all methods, and hence usable (meaningful) in all the methods in the program
4.3.9	Construct algorithms using loops, branching.	3	Teachers must ensure algorithms use the symbols from the approved notation sheet.	Loops - in Pseudocode : loop C from 1 to 10 ... end loop

			LINK Approved notation sheet. **MYP** Mathematics: using flow charts to solve problems in real-life contexts, logic, algorithms **MYP** Technology: design cycle (inputs, processes, outputs, feedback, iteration). **LINK** Connecting computational thinking and program design.	loop while not FOUND do … end loop Branching - in Pseucode : if … then … else if … then … else ...

4.3.10	Describe the characteristics and applications of a collection.	2	Characteristics:
			● Contains similar elements.
			LINK HL extension, recursive thinking.
			LINK General principles of computational thinking, connecting computational thinking and program design.
				A COLLECTION is like a linked-list,
				but the order of elements is not guaranteed.
				Collection methods in Pseudocode are:
				.addItem(new data item)
				.resetNext( ) start at beginning of list
				.hasNext( ) checks whether there are still more items in the list
				.getNext( ) retrieve the next item in the list
				.isEmpty( ) check whether the list is empty
				
				Example:
				
				`NAMES = new Collection()`
				
				`NAMES.addItem("Bob")`
				`NAMES.addItem("Dave")`
				`NAMES.addItem("Betty")`
				`NAMES.addItem("Kim")`
				`NAMES.addItem("Debbie")`
				`NAMES.addItem("Lucy")`

				NAMES.resetNext()
				
				output "These names start with D"
				
				loop while NAMES.hasNext()
				  NAME = NAMES.getNext()
				  if firstLetter(NAME) = "D" then
				  output NAME
				  end if
				end loop
				
				method firstLetter(s)
				  return s.substring(0,1)
				end method
				
				Output:
				Dave
				Debbie

4.3.11	Construct algorithms using the access methods of a collection.	3	**LINK** Connecting computational thinking and program design.	See the examples in **Pseudocode Practice Tool**
4.3.12	Discuss the need for sub-programmes and collections within programmed solutions.	3	Show an understanding of the usefulness of reusable code and program organization for the individual programmer, team members and future maintenance. **LINK** General principles of computational thinking, connecting computational thinking and program design.	Too bad that IB Pseudocode contains no facility for writing methods or sub-programs.

			MYP Technology: use of software such as Alice.	
4.3.13	Construct algorithms using pre-defined sub-programmes, one-dimensional arrays and/or collections.	3	**MYP** Mathematics: using flow charts to solve problems in real-life contexts, logic, algorithms. **MYP** Technology: design cycle (inputs, processes, outputs, feedback, iteration); use of software such as Alice. Students will only be required to analyse flow charts in the externally assessed components. Students will be expected to write and analyse pseudocode in the externally assessed components. **S/E**, **AIM 8** Appreciate the implications of using available code from sources such as online forums. **LINK** Connecting computational thinking and program design.	See the examples in **Pseudocode Practice Tool**