3 Subject content

3.1 Algorithms

3.1.1 Representing algorithms

Content	Additional information
Understand and explain the term algorithm.	An algorithm is a sequence of steps that can be followed to complete a task. Be aware that a computer program is an implementation of an algorithm and that an algorithm is not a computer program.
Understand and explain the term decomposition.	Decomposition means breaking a problem into a number of sub problems, so that each sub problem accomplishes an identifiable task, which might itself be further subdivided.
Understand and explain the term abstraction.	Abstraction is the process of removing unnecessary detail from a problem.
Use a systematic approach to problem solving and algorithm creation representing those algorithms using pseudocode and flowcharts.	Any exam question where students are given pseudocode will use the Oxford International AQA standard version. However, when students are writing their own pseudocode they may do so using any form as long as the meaning is clear and unambiguous.
Explain simple algorithms in terms of their inputs, processing and outputs.	Students must be able to identify where inputs, processing and outputs are taking place within an algorithm.
Determine the purpose of simple algorithms.	Students should be able to use trace tables and visual inspection to determine how simple algorithms work and what their purpose is.

3.1.2 Efficiency of algorithms

Content	Additional information
Understand that more than one algorithm can be used to solve the same problem.	
Compare the efficiency of algorithms explaining how some algorithms are more efficient than others in solving the same problem.	Formal comparisons of algorithmic efficiency are not required. Exam questions in this area will only refer to time efficiency.

3.1.3 Searching algorithms

Content	Additional information
Understand and explain how the linear search algorithm works.	Students should know the mechanics of the algorithm and be able to follow and write pseudo-code for it.
Understand and explain how the binary search algorithm works.	Students should know the mechanics of the iterative version of the algorithm and be able to follow and write pseudocode for it.
Compare and contrast linear and binary search algorithms.	Students should know the advantages and disadvantages of both algorithms.

3.1.4 Sorting algorithms

Content	Additional information
Understand and explain how the merge sort algorithm works.	Students should know the mechanics of recursive version of the algorithm. It will be sufficient for students to explain this algorithm in prose; they will not be expected to be able to write pseudocode for it. Students should be able to demonstrate how a merge sort would be performed on a given set of data.
Understand and explain how the bubble sort algorithm works.	Students should know the mechanics of the algorithm and be able to follow and write pseudo-code for it. Students will be expected to know the version of the algorithm that uses two nested loops, with the outer loop being indefinitely controlled by a condition that tests if any swaps were made and the inner loop being controlled definitely.
Compare and contrast merge sort and bubble sort algorithms.	Students should know the advantages and disadvantages of both algorithms.

3.2 Programming

Students need a theoretical understanding of all the topics in this section for the exams even if the programming language(s) they have been taught do not support all of the topics. Written exams will always present algorithms and code segments using the OxfordAQA' pseudocode, which can be found in the teaching guidance on the OxfordAQA website, although students can present their answers to questions in any suitable format and do not need to use the pseudocode when answering questions.

3.2.1 Data types

Content	Additional information
Understand the concept of a data type. Understand and use the following appropriately: • integer • real • Boolean • character • string.	Depending on the actual programming language(s) being used by the students, these data types may have other names. For example real numbers may be described as float. In exams we will use the general names given in this specification.

3.2.2 Programming concepts

Content	Additional information
Use, understand and know how the following statement types can be combined in programs: • variable declaration • constant declaration • assignment • iteration • selection • subroutine (procedure/function).	The three combining principles (sequence, iteration/ repetition and selection/choice) are basic to all imperative programming languages. Students should be able to write programs using these statement types. They should be able to interpret algorithms that include these statement types. Students should know why named constants and variables are used.

Content	Additional information

Use definite and indefinite iteration, including indefinite iteration with the condition(s) at the start or the end of the iterative structure.	A theoretical understanding of condition(s) at either end of an iterative structure is required, regardless of whether or not they are supported by the language(s) being used.
	
	An example of definite iteration would be:
	
	`FOR i←1 TO 5`
	`  … Instructions here …`
	`ENDFOR`
	
	An example of indefinite iteration with the condition at the start would be:
	
	`WHILE NotSolved`
	`  … Instructions here …`
	`ENDWHILE`
	
	An example of indefinite iteration with the condition at the end would be:
	
	`REPEAT`
	`  … Instructions here …`
	`UNTIL Solved`

Use nested selection and nested iteration structures.	An example of nested iteration would be:
	
	`WHILE NotSolved`
	`  … Instructions here …`
	`  FOR i←1 TO 5`
	`    … Instructions here …`
	`  ENDFOR`
	`  … Instructions here …`
	`ENDWHILE`
	
	An example of nested selection would be:
	
	`IF GameWon THEN`
	`  … Instructions here …`
	`  IF Score > HighScore THEN`
	`    … Instructions here …`
	`  ENDIF`
	`  … Instructions here …`
	`ENDIF`

Use meaningful identifier names and know why it is important to use them.	Identifier names include names for variables, constants and subroutine names.

3.2.3 Arithmetic operations in a programming language

Content	Additional information
Be familiar with and be able to use: • addition • subtraction • multiplication • real division • integer division, including remainders.	Integer division, including remainders is usually a two-stage process and uses' modular arithmetic: eg the calculation 11/2 would generate the following values: Integer division: the integer quotient of 11 divided by 2: (11 DIV 2) = 5 Remainder: the remainder when 11 is divided by 2: (11 MOD 2) = 1

3.2.4 Relational operations in a programming language

Content	Additional information
Be familiar with and be able to use: • equal to • not equal to • less than • greater than • less than or equal to • greater than or equal to. In assessment material we will use the following symbols: =, ≠, <, >, ≤, ≥	Students should be able to use these operators within their own programs and be able to interpret them when used within algorithms. Note that different languages may use different symbols to represent these operators.

3.2.5 Boolean operations in a programming language

Content	Additional information
Be familiar with and be able to use: • NOT • AND • OR.	Students should be able to use these operators, and combinations of these operators, within conditions for iterative and selection structures.

3.2.6 Data structures

Content	Additional information
Understand the concept of data structures.	It may be helpful to set the concept of a data structure in various contexts that students may already be familiar with. It may also be helpful to suggest/demonstrate how data structures could be used in a practical setting.
Use arrays (or equivalent) in the design of solutions to simple problems.	Only one and two dimensional arrays are required. In Python, a list is a suitable alternative to an array for this specification.

Use records (or equivalent) in the design of solutions to simple problems.	In C#, structs can be used to create records.
	
	In Python, classes can be used in a non-object-oriented way to create records. For example:
	
	`class Coordinate():`
	`  def __init__(self):`
	`  self.x = 0`
	`  self.y = 0`
	
	`myposition = Coordinate()`
	
	`myposition.x = 10`
	`myposition.y = 5`
	
	In Visual Basic, structures can be used to create records.

3.2.7 Input/output and file handling

Content	Additional information
Be able to obtain user input from a keyboard.	
Be able to output data and information from a program to the computer display.	
Be able to read/write from/to a text file.	

3.2.8 String handling operations in a programming language

Content	Additional information
Understand and be able to use: • length • position • substring • concatenation • convert character to character code • convert character code to character • string conversion operations.	Expected string conversion operations: • string to integer • string to real • integer to string • real to string.

3.2.9 Random number generation in a programming language

Content	Additional information
Be able to use random number generation.	Students will be expected to use random number generation within their computer programs. An understanding of how pseudo-random numbers are generated is not required.

3.2.10 Subroutines (procedures and functions)

Content	Additional information
Understand the concept of subroutines.	Know that a subroutine is a named 'out of line' block of code that may be executed (called) by simply writing its name in a program statement.
Explain the advantages of using subroutines in programs.	
Use and describe the use of parameters to pass data within programs.	Students should be able to use subroutines that require more than one parameter. Students should be able to describe how data is passed to a subroutine using parameters.
Use subroutines that return values to the calling routine.	Students should be able to describe how data is passed out of a subroutine using return values.

3.2.11 Structured programming

Content	Additional information
Describe the structured approach to programming.	Students should be able to describe the structured approach including modularised programming, clear, well documented interfaces (local variables, parameters) and return values. Teachers should be aware that the terms 'arguments' and 'parameters' are sometimes used but in examinable material we will use the term 'parameter' to refer to both of these.
Explain the advantages of the structured approach.	

3.2.12 Robust and secure programming

Content	Additional information
Be able to write simple data validation routines.	Students should be able to use data validation techniques to write simple routines that check the validity of data being entered by a user. The following validation checks are examples of simple data validation routines: • checking if an entered string has a minimum length • checking if a string is empty • checking if numeric data entered lies within a given range (eg between 1 and 10).
Be able to write simple authentication routines.	Students should be able to write a simple authentication routine that uses a username and password. Students will be required to use only plain text usernames and passwords (ie students will not need to encrypt the passwords).
Be able to select suitable test data that covers normal (typical), boundary and erroneous data. Be able to justify the choice of test data.	