OXFORD
INTERNATIONAL
AQA EXAMINATIONS

INTERNATIONAL GCSE COMPUTER SCIENCE

(9210)

Outline schemes of work

For teaching from September 2017 onwards

For International GCSE exams in June 2019 onwards

This scheme of work suggests possible teaching and learning activities for each section of the specification. There are more activities suggested than it would be possible to teach. It is intended that teachers should select activities appropriate to their students and the curriculum time available. The first two columns summarise the specification references, whilst the learning outcomes indicate what most students should be able to achieve after the work is completed. The Resources column indicates resources commonly available to schools, and other references that may be helpful. The timings are only suggested, as are the possible teaching and learning activities, which include references to experimental work. Resources are only given in brief and risk assessments should be carried out.

As most of the resources suggested in this scheme of work have not been created specifically for this specification it is important that teachers check the exact requirements of the specification to ensure that all aspects of it are covered by their students.

General timings

The scheme of work is based on a total of 120 guided learning hours. Of these 120 hours, it is recommended that:

- Approximately 50 hours are used for teaching specification sections 3.1 and 3.2 (algorithms and programming).
- Approximately 30 hours are used for consolidation or extension of programming skills.
- Approximately 30 hours are used for teaching specification sections 3.3 to 3.8.
- Approximately 10 hours are reserved for assessments and exam preparation, including preparing for the on-screen programming exam using the skeleton program pre-release.

The table below summarises the recommended time allocations for specification sections 3.1 to 3.8:

Spec reference	Topic	Learning time (Hours)
3.1	Algorithms	50
3.2	Programming	
3.1 – 3.2	Programming consolidation/extension	30
3.3	Data representation	8.5
3.4	Computer systems	6
3.5	Computer networks	4
3.6	Cyber security	2.5
3.7	Relational Databases and SQL	5.5
3.8	Web page design	4.25
3.1 – 2.8	Assessments and exam preparation	9.25
	Total	**120**

As programming and algorithms are at the core of the subject, it is recommended that these topics are taught throughout the course, with the other topics taught alongside them. Based on this scheme of work, in a typical week, approximately three quarters of the time would be spent programming.

General resources

The following resources may be of general use whilst teaching the specification. Specific resources for individual topics are recommended in the scheme of work.

- **BBC Bitesize GCSE Computer Science**: Notes, tests and some videos on many topics covered by this specification: **bbc.co.uk/education/subjects/z34k7ty**
- **Computing at schools**: Community for UK based computer science teachers, with discussion forums and many useful teaching resources that have been shared (free subscription required): **community.computingatschool.org.uk/**
- **AQA GCSE teaching guides**: A collection of guides written to help teachers deliver the content in the AQA GCSE Computer Science specification in the UK, many of which can be directly used for this specification: **aqa.org.uk/subjects/computer-science-and-it/gcse/computer-science-8520/teaching-resources**
- **CS unplugged**: A collection of resources that teach principles of computer science in fun ways: **csunplugged.org/**

There are also many websites designed to help students to learn to program. A few examples are:

- Code Academy: **codecademy.com/**
- Khan Academy (Computer programming): **khanacademy.org/computing/computer-programming**
- Learn Python: **learnpython.org/**
- Code.org: **code.org/learn**

3.1 Algorithms and 3.2 Programming

Total time for teaching this section: 50 hours.

Many different approaches can be taken to helping students to understand algorithms and to develop the skills required to be able to write computer programs. In the approach outlined here, the two topics are taught together. The exercises suggested are grouped together so that students are able to apply the same techniques to solve similar problems before moving on to learning new techniques.

The problems within each group are graduated in difficulty. At the start of tackling a group of exercises it would be helpful to give students some program code that they have to modify so that they become familiar with the basic syntax required in their language. Later on, more emphasis should be placed on students designing the solutions themselves. For example, algorithms could be presented in pseudocode or as flowcharts that students need to implement by writing code in their own programming language. Eventually, students should just be set problems where they have to make all the design decisions necessary to produce a fully working program from a problem description.

Students only need to be able to write programs in console mode for this course, but there is time for students to do some programming of applications with a graphical user interface if this is desired.

In the exam, students will need to be able to understand algorithms written in the Oxford AQA Exams pseudocode, so it is important that students get to see and use this as they learn to program. A guide to the Oxford AQA Exams pseudocode is available.

Note that for this section of the specification, in the summary of specification content column only content that is being covered for the first time or that is the focus of the exercises is listed. In the later programming exercises, skills introduced earlier will naturally be revisited and reinforced.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.1.1, 3.2.1, 3.2.2, 3.2.3, 3.2.7	Understand and use string, integer and real data types appropriately. Understand how variable declaration and assignment can be used in programs. Be able to use addition, subtraction, multiplication and real division. Be able to perform input and output. Use meaningful identifier names and know why it is important to	Apply the listed programming techniques. Choose appropriate data types. Use meaningful identifier names and know why. Understand what an algorithm is and the difference between an algorithm and program.	3 hours	Students should be introduced to basic input and output commands, declaring variables (if required by language), and using arithmetic operations. Students will also need to be taught basic aspects of the IDE for their programming language, eg how to run a program, how to load/save, how error messages are presented and what they mean. Students should be introduced to the idea of an algorithm and that a program is an implementation of an algorithm. **Exercises could include:** Getting the computer to display "Hello World". Getting the user to type in	Notes and videos introducing algorithms and example uses: **bbc.co.uk/education/guides/z22wwmn/revision** Notes on variables and data types (and some other concepts not required until later): **bbc.co.uk/education/guides/zc6s4wx/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	use them. Understand and explain the term algorithm.			their name and outputting hello to them (possibly concatenating forename and surname input separately). Doing simple calculations, for example adding three numbers, multiplying two numbers together. Doing more complex calculations, for example area of a rectangle, area of a triangle, area of a circle, area of a trapezium.		
3.2.2, 3.2.4, 3.2.5, 3.2.12	Be able to use selection (if, else, else if, case/switch if appropriate). Be able to use a range of relational operators. Be familiar with	Apply the programming techniques listed. Choose appropriate test data to use to check their programs.	3 hours	The focus in this section is on the use of selection statements to determine the path of code execution. Exercises should build in difficulty, starting with simple Yes/No answers using just an If statement then building in complexity in terms of the number of possible outcomes and the	Notes and video on use of selection statements: **bbc.co.uk/education/guides/zrxncdm/revision/3** Notes on use of AND, OR and NOT: **bbc.co.uk/education/guides/zc4bb9q/r**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	and able to use NOT, AND, OR. Using nested selection structures. Be able to select suitable test data that covers normal (typical), boundary and erroneous data. Be able to justify the choice of test data. Be able to understand pseudocode and flowcharts.			complexity of the criteria used. Psuedocode and flowcharts could be used to illustrate some algorithms which students could then write program code for. Whilst completing these exercises, consideration should be given to choosing test data, which is particularly important in boundary situations of which there are many in these exercises. **Exercises could include:** Exam mark pass/fail. Determining if a person is a child/adult/pensioner based on their age. Allocating an exam grade based on mark ranges.	**evision/4**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				Identifying the biggest of two or three numbers. Identifying if a triangle is scalene, isosceles or equilateral. Classifying the temperature based on a range eg 0 or below = freezing, above 0 but 10 or below = warm.		
3.2.2	Be able to use definite iteration. Be able to use nested iteration.	Be able to use definite iteration.	4 hours	Students should be introduced to the concept of definite iteration and a loop counter. Pseudocode and flowcharts could be used to illustrate algorithms. **Exercises could include:** Counting from one to ten. Displaying a times table, or all times tables. Adding up five numbers. Average the same numbers and identify the highest and	Notes and videos on use of iteration: **bbc.co.uk/education/guides/zrxncdm/revision/4**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				lowest. Working out factors of a number using brute-force approach. Identifying prime numbers using brute force approach.		
3.1.1, 3.2.2, 3.2.8, 3.2.9, 3.2.12	Be able to use indefinite iteration with conditions at start and end of loop. Be able to use random number generation. Be able to use some string handling techniques. Be able to write simple data validation routines.	Apply the programming techniques listed. Be able to write simple authentication and validation routines. Understand what abstraction is.	4 hours	Students need to be taught about indefinite iteration and how to use this in their programming language. For students using Python which does not have a post-conditioned loop, they should be taught how to implement post-conditioned loops as equivalent pre-conditioned loops. Students also need to know how to express these types of loop as pseudocode and flowcharts. As students are now starting to tackle more complex problems, the	Notes and videos on use of iteration: **bbc.co.uk/education/guides/zrxncdm/revision/4** Video on abstraction: **bbc.co.uk/education/guides/zttrcdm/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	Be able to write simple authentication routines. Understand and explain the term abstraction.			concept of abstraction, ie removing unnecessary details from a problem could be introduced at this point. **Exercises could include:** Performing simple validation, eg that a typed value falls within a range or that an entered value cannot be left blank or is shorter than a minimum length. Add up a sequence of numbers of unknown length. Ask users to enter a password until the correct password is entered, displaying suitable messages. Guessing game where players have to guess randomly chosen number		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				until they guess correctly, with clues given about whether guess is too high/low. Keep rolling two dice until a double six is scored, counting how many goes this takes. Darts game where darts are thrown and get a random score on board. Game starts at a total and plays with the total decreased by each dart thrown until 0 is achieved, eg 501.		
3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.2.6	Understand the concept of data structures. Use one-dimensional arrays (or equivalent) in the design of solutions to simple problems.	Be able to use one-dimensional arrays. Understand the searching and sorting algorithms listed and be able to compare their efficiency.	5 hours	Students need to be introduced to the concept of a one-dimensional array and should have the opportunity to solve problems using them. They should also cover the four searching and sorting algorithms and have the opportunity to code all of them except the merge sort.	Notes on data structures and arrays: **bbc.co.uk/education/guides/z4tf9j6/revision/1** Video on searching algorithms: **bbc.co.uk/education/clips/zpbyxsg**	Questions will use arrays with a first index of 0 (not 1).

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	Understand that more than one algorithm can be used to solve the same problem. Compare the efficiency of algorithms. Understand and explain how linear and binary search algorithms work and compare them. Understand and explain how bubble and merge sort algorithms work and compare them. Use trace tables.			Before coding these algorithms, it would be helpful for students to look at them in pseudocode and to trace their execution in a trace table to ensure that they understand how they function. **Exercises could include:** Input a list of names (or something else) and redisplay them. Input a list of parcel weights. Total the weights and work out the average, lowest and highest weight. Warn if any parcel exceeds a maximum weight. Calculate cost of sending the parcels. Search a dictionary to check if a word is in it using the linear search method. Improve the dictionary	Video on sorting algorithms: **bbc.co.uk/education/clips/zt9rnbk** Video of bubble sort using Lego bricks: **youtube.com/watch?v=MtcrEhrt_K0** Video on bubble sort: **youtube.com/watch?v=8Kp-8OGwphY** Simple explanation of Merge Sort video: **youtube.com/watch?v=EeQ8pwjQxTM** Video comparing linear and binary search: **youtube.com/watch?v=JQhciTuD3E8**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				program to use the binary search method. Use the bubble sort algorithm to sort data (eg names) in an array. Look theoretically at how the merge sort algorithm would perform the same sort (implementing merge sort is beyond GCSE but more able students could attempt this). Compare the efficiency of the search and sort algorithms. Represent a game of snakes and ladders using a one-dimensional array to indicate the positions of snakes and ladders.		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.1.1, 3.2.2, 3.2.3, 3.2.10, 3.2.11	Understand and explain the term decomposition. Describe the structured approach to programming. Explain the advantages of the structured approach. Understand the concept of subroutines and be able to use them in programs, including the use of local variables. Explain the advantages of using subroutines in programs.	Be able to decompose problems into subroutines and call them and know why this is a good idea. Be able to perform integer division, including the use of remainders.	3 hours	Students should be taught about why, when writing longer programs, it is useful to decompose them, and the facilities in their programming language to do this. They should also cover the difference between local and global variables. At this stage, parameters and return values can be ignored. **Exercises could include:** Make a maths toolkit, with a menu which is used to call different subroutines to work out (for example) the area of different shapes. Make a program that will allow conversion of numbers between different number bases, with different functions being used for different conversions eg binary to	Video explaining the concept of decomposition: **bbc.co.uk/education/clips/z8nx82p** Notes and video on decomposition, including the use of parameters (which is not required until later): **bbc.co.uk/education/guides/z9hykqt/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	Integer division, including remainders.			decimal. In all subsequent programs, students should be encouraged to consider how the programs can be decomposed into functions.		
3.2.1, 3.2.8, 3.2.11	Use a structured approach to programming, in particular focusing on the use of parameters and return values. Use a range of string handling operations. Use the char and Boolean data types.	Be able to use well-defined interfaces to subroutines, using parameters and return values. Use string handing operations and the char data type.	6 hours	Emphasis should be on passing input to the functions as parameters and using return to pass values back to the calling program. Input/output via the keyboard/screen should not happen within the functions. Students should be taught why this is important, for example in terms of being able to develop and test modules independently and reuse code.	Notes and video on decomposition: **bbc.co.uk/education/guides/z9hykqt/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				Exercises could include: Develop a function that returns the highest of two numbers. Adapt this to find the highest of three numbers or to perform other mathematical operations. Develop a function that indicates if a number is even or not. Develop a function that works out n factorial (n!). Develop a function that returns a string that has been encrypted using the Caesar Cipher with a key selected by the user. Add a decryption function. Develop a function to convert a string into Morse code. Develop a function that will return a true/false value,		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				indicating if two words are anagrams of each other. Develop a function that, when sent a number will return a true/false value indicating if the number is a perfect number or not. Use this in a program to search for perfect numbers using brute force. In all subsequent programs, students should be encouraged to consider how the programs can be decomposed into functions with interfaces that use parameters and return values.		
3.2.7	Be able to read/write from/to a text file.	Be able to use text files for permanent storage of data.	4 hours	Students first need to understand what a text file is. They should then modify some of the previous programs that they have written to read input from/ save output to a text file.		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				Some suitable programs to modify would be: Snakes and ladder positions could be stored in a text file (allowing for the possibility of different boards). The dictionary that is searched could be stored in a text file. The sorting program could load the list to sort from a text file and save the sorted list to a different text file. Students should consider how to deal with possible issues such as saving over an existing file or being asked to load a file that does not exist. This would be a point where exception handling could be considered.		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.2.2, 3.2.6	Use two-dimensional arrays (or equivalent) in the design of solutions to simple problems. Use nested iteration. Use of constants.	Use two-dimensional arrays, nested iteration and constants.	10 hours	Students should have the opportunity to write programs using two-dimensional arrays. They will need to consider/design how the arrays can be used to represent the problem. Data stored in a two-dimensional array is usually displayed most conveniently using nested loops. A range of game can be readily implemented using two-dimensional arrays. If constants have not yet been encountered by students, they could be introduced here to, for example, store the size of a game board. **Exercises could include:** Snakes and ladders. Noughts and crosses.	Brief notes on two-dimensional arrays: **bbc.co.uk/education/guides/z4tf9j6/revision/3**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				Battleships.		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.2.6, 3.2.12	Use records (or equivalent) in the design of solutions to simple problems. Be able to write simple authentication routines.	Use records. Be able to write simple authentication routines.	4 hours	Students should be introduced to the concept of records and why logically grouping related data together is a sensible approach. **Exercises could include:** Adapt the dictionary program that was written earlier to store equivalent words in two languages in an array of records and perform translation between them. Write an address book program, or a program to keep track of any other data. This data could be saved/loaded from a text file using CSV format. Write a login system with usernames and passwords stored in a file and then loaded into an array of records.		Students using Python can use different methods for representing records. The approach that will be used in the programming exam is detailed in section 3.2.6 of the specification.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.1.1	Use a systematic approach to problem solving and algorithm creation representing those algorithms using pseudocode and flowcharts. Explain simple algorithms in terms of their inputs, processing and outputs. Determine the purpose of simple algorithms.	Students can understand algorithms expressed in pseudocode and flowcharts and use these methods to write algorithms. They can trace the execution of algorithms using a trace table and identify the purpose of an algorithm. They can identify the inputs and outputs of an algorithm together with the required processing,	2 hours	Throughout learning to program, students should be exposed to how algorithms can be expressed using pseudocode or flowcharts. Students need to have some practice at being able to understand and write algorithms using these methods. They also need to be able to use trace tables to record the values of variables as an algorithm is stepped through and to be able to identify the purpose of an algorithm by tracing it. These skills will be assessed in the exam. It is useful to teach them in parallel with leaning to program (perhaps as homework exercises) but it could also be worth giving		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				students the opportunity to consolidate their ability to apply these skills. Students should complete exercises where they have to read and write pseudocode and flowcharts, complete trace tables and deduce the purpose of algorithms.		
3.2.13	Know that there are different levels of programming language: low-level, high-level and explain the main differences between them. Know that machine code and assembly language are considered to be low-level	Students should understand the differences between low and high-level languages. They should know the differences between machine code and assembly language. They should know that all programs must be converted to machine code before they can be executed.	2 hours	Students only need a theoretical understanding of this topic, so this topic would be best delivered by the teacher as a presentation or through notes or a video with students given the opportunity to answer questions on it. Including real examples of assembly language and machine code is helpful. Students could be given some opportunity to write	Online notes covering some of these topics: **bbc.co.uk/education/guides/zgmpr82/revision/1** An assembly language tutorial video for a raspberry Pi: **youtube.com/watch?v=ViNnfoE56V8** Video comparing C code with equivalent	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	languages and explain the differences between them. Understand that ultimately all programming code written in high-level or assembly languages must be translated into machine code. Understand that machine code is expressed in binary and is specific to a processor or family of processors. Understand the advantages and disadvantages of low-level	Students should understand the advantages and disadvantages of programming in high-level and assembly languages. Students should know the purpose of and be able to compare the different types of translator.		very simple programs in assembly language so that they can see how assembly language compares to a high-level language but this is not a requirement. Students using Visual Basic or C# could find the exe file that is output when they compile a program. Students could look at an assembly language instruction set (eg ARM) to consider the types of instruction available and their limitations.	assembly language code: **youtube.com/watch?v=yOyaJXpAYZQ** Video on translation software: **youtube.com/watch?v=1OukpDfsuXE**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	language programming compared with high-level language programming. Understand that there are three common types of program translator: interpreter, compiler, assembler. Explain the main differences between these three types of translator and understand when it would be appropriate to use each type of translator.					

3.1 and 3.2 Programming consolidation/extension

Total time for teaching this section: 30 hours.

Once students have developed their programming skills it is important that they get the opportunity to consolidate them by working on other programming projects. These could be set by the teacher for the whole class or individually chosen to reflect students' interests and ability.

Past and sample coursework (NEA) assignments set for GCSE coursework in England are one source of ideas for consolidation programming tasks. Some assignments can be accessed at **aqa.org.uk/subjects/computer-science-and-it/gcse/computer-science-8520/assessment-resources**.

When completing consolidation tasks, it is important that students get to develop their own skills in analysing problems and designing and testing their solutions, as well as coding them.

Able students could also be given the opportunity to extend their skills, for example:

- If a student learnt how to program in console mode they could be given the opportunity to develop applications with a graphical user interface.
- A student could learn how to access a database using SQL from their programming language, linking topics 3.2 and 3.7.

A student could try to do some client or server-side scripting, linking topics 3.2 and 3.8.

3.3 Data representation

Total time for teaching this section: 8.5 hours.

The topics in this section of the specification all require students to apply their skills, so it is important that they get plenty of opportunities to do this.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.3.1, 3.3.2	Explain why computers use binary. Understand how binary can be used to represent whole numbers and be able to convert between binary and decimal and vice-versa.	Understand that computers use binary to represent data and instructions. Be able to convert between binary and decimal and vice-versa.	1 hour	Look at how computers store data conceptually as on and off states and how this can be conceived numerically as binary (may be easier to look at early computers with valves, transistors). Review how the decimal system works with 10 digits and place values that are powers of 10 and relate this to how binary works with 2 digits and place values that are powers of 2. Show how a binary number can be converted to decimal by adding the place values of columns with 1s in.	Notes on binary and conversion: **bbc.co.uk/education/guides/ztcdtfr/revision/1** Binary conversions game: **learningnetwork.cisco.com/docs/DOC-1803**	Exam questions will only use values up to 8 bits in length.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				Show how decimal can be converted to binary by working from left to right. Consider the highest and lowest decimal value that can be stored in 8 bits. This is a topic that students must practice so they need to complete conversion exercises, possibly some in class and some for homework.		
3.3.1, 3.3.2	Understand how hexadecimal can be used to represent whole numbers and be able to convert between decimal and hexadecimal and binary and hexadecimal. Understand why hexadecimal is	Be able to convert between decimal and hexadecimal and binary and hexadecimal. Understand why hexadecimal is often used in computer science and give examples of where it is used.	1 hour	Consider why binary is not easy for humans to use (eg long strings of digits, easy to transpose, hard to remember). Explain why hexadecimal is a good shorthand for binary (4 bits = 1 hex digit) and look at where hex is used eg colour codes, MAC addresses, memory editors.	Online notes on conversions: **bbc.co.uk/education/guides/zp73wmn/revision**	Exam questions will only use 8-bit values.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	often used in computer science.			Look at methods for converting between decimal and hexadecimal and vice-versa (remember only 8-bit numbers are needed). Look at the quick method for converting between binary and hexadecimal and vice-versa in groups of 4 bits. Students needs to complete plenty example conversion exercises in class and for homework.		
3.3.3	Know the units that are used to measure quantities of bytes.	Students know the units bit, byte, kilobyte, megabyte, gigabyte and terabyte.	0.5 hours	Explain the names of the measurements used for quantities. Consider a comparison with measurements for distance where different but related measurements are used depending on the magnitude of the distance being measured (eg cm, m, km).	Reference for units: **wikipedia.org/wiki/Units_of_information**	Students need to remember that for this specification powers of 10 are used for units not powers of 2.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				Emphasise that this specification uses the SI definitions of the units which are powers of 10, but refer to the historical definitions using powers of 2 which students may be familiar with. Look at measurements of sizes of typical things eg RAM in a computer, size of a hard disk, download allowances. Students could be set some exercises working out file sizes or converting between units.		
3.3.4	Be able to add together two binary numbers. Be able to perform logical shifts.	Be able to add together two binary numbers. Be able to perform logical shifts.	0.5 hours	Students need to be shown the method for completing binary addition of two numbers, including how to deal with carries. Then they should complete some exercises to practice this.	Notes on binary addition (note that negative numbers are not required for this spec): **bbc.co.uk/education/guides/zjfgjxs/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				Students should then be shown how a binary shift can be used to double/ approximately halve a number.	Binary addition video: **youtube.com/watch?v=ypqYoFbPfTk**	
3.3.5	Understand character sets including ASCII and Unicode and the advantages of Unicode.	Understand character sets including ASCII and Unicode and the advantages of Unicode. Be able to use a character table to convert a message from binary to a character set and vice-versa.	1 hour	Look at the ASCII table. Complete exercise converting a message from binary to characters and vice-versa. Note how similar characters are in blocks eg all capital letters. Consider limitations of ASCII (limited number of characters) and look at how Unicode solves these. This topic could be linked to programming through the use of the programming language commands for conversion between character codes	Online notes on character sets: **bbc.co.uk/education/guides/zp73wmn/revision/5** The ASCII table: **asciitable.com/** Official Unicode website: **unicode.org/** Unicode character tables: **unicode.org/charts/**	Students need to be aware that similar characters run in blocks eg A = 65, B = 66 etc.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				and characters.		
3.3.6	Understand how images can be represented as bitmaps, including key terms. Be able to calculate file sizes. Be able to convert between binary and image data.	Define key terms eg pixel, colour depth. Calculate file sizes for bitmap images. Convert between an image and binary and vice-versa.	1.5 hours	Look at bitmap images using a graphics package, use zoom to identify pixels and colours (possible link to hex). Introduce colour depth by considering how different patterns of 0s and 1s could be used to represent colours. A colour depth of n bits allows 2^n colours. Perform some exercises where students have to convert small images between images and binary data and vice-versa. Start with black and white images and then images with small numbers of colours. Explain how to calculate the size of an image file and then students	A bitmap image editor eg Paint. Online notes and test (note: vector graphics are not required for specification): **bbc.co.uk/education/guides/zqyrq6f/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				complete some sample calculations.		
3.3.7	Understand analogue sound must be converted to digital form for storage. Describe how sound is represented using sample rate and sample resolution. Calculate sound file sizes. .	Understand the difference between analogue and digital. Be able to recognise analogue and digital quantities. Be able to explain how sample rate and sample resolution affect sample quality and file size. Be able to calculate file sizes for sound files.	1.5 hours	Discuss difference between analogue and digital quantities. Look at how sound can be represented electronically as a waveform – a package such as Audacity can be used to allow students to look at sounds and record their own. Use a graph to show how the sampling process works and how sample quality and size would be affected by changing sample rate and sample resolution. Perform calculations of sound file sizes. Students should carry out exercises that involve	A sound editing package such as Audacity **audacityteam.org/**. Video (goes beyond GCSE level): **youtube.com/watch?v=HqHIOA-Fcuw**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				identifying analogue and digital quantities, converting between an analogue waveform and digital samples and calculating sound file sizes.		
3.3.8	Explain what data compression is, and why it is used. Be able to compress/ decompress data using Huffman coding and calculate how many bits are saved. Be able to compressed/ decompress data using RLE.	Explain what data compression is, and why it is used. Be able to compress/decompress data using Huffman coding and calculate how many bits are saved. Be able to compressed/decompress data using RLE. .	1.5 hours	Students could try creating ZIP files or comparing the size of JPEG (compressed) and Bitmap files of the same image to see the effect of compression. In discussion, consider why compression is useful – either in the context of transmission or storage of data eg faster downloads, more photos on memory cards etc. RLE is the simplest of the two techniques so best to cover this first. Look at how it can be used with	Compression methods and RLE video: **youtube.com/watch?v=pbhe7DXwcIQ** RLE video: **youtube.com/watch?v=ypdNscvym_E** Huffman coding video: **youtube.com/watch?v=apcCVfXfcqE** .	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				small images and get students to try compressing small bitmaps using it. Consider why it is not suitable for some images and many types of data. Look at Huffman coding and the concept of variable-length codes with more common characters having shorter codes. Students should try to use a Huffman tree to decode some text stored as binary data. At this point, a calculation of how much memory was saved compared to using 7-bit ASCII can be made. Then introduce a technique that can be used to build a Huffman tree for a given piece of		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				text. Different techniques can be used but they generally involve repeatedly combining the two characters with the least frequently occurring letters to build a tree. Students need to complete practice exercises compressing and decompressing data using both RLE and Huffman coding and calculating how much memory is saved when Huffman coding is used. There are lots of videos available illustrating these techniques.		

3.4 Computer Systems

Total time for teaching this section: 6 hours

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.4.2	Construct truth tables for NOT, AND, OR gates, Construct truth tables for simple logic circuits and interpret them. Create simple logic circuit diagrams. Be able to write Boolean expressions for logic circuits and vice-versa.	Be able to construct truth tables for gates and circuits. Be able to draw logic circuits to represent a simple logic problem. Be able to write Boolean expressions for logic circuits and vice-versa.	2 hours	Consider the basic operations of AND, OR and NOT (students may have already come across these in the context of programming or databases depending on the order in which the sections are taught). Look at truth tables for each gate. Draw a logic circuit and then build a truth table for it. Students should then try some exercises completing truth tables for different logic circuits. Introduce the idea of drawing a logic circuit to represent a specific	Online logic gate simulator: **academo.org/demos/logic-gate-simulator/** Online logic gate simulator: **expertgear.com/projects/boolLogicSim/** Notes and video on logic circuits and Boolean expressions (note XOT not required for spec): **bbc.co.uk/education/guides/zc4bb9q/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				problem. Students should then try to draw logic circuits for a few problems. This could be done on paper or using an online logic circuit simulator or physically using electronics or tools such as logic goats. Introduce Boolean notation and explain how logic problem can be described unambiguously using it. Students should then try out exercises where they design a circuit for a Boolean expression and vice-versa. It is not required for the specification but it would be useful to link this in to hardware and the design of the processor by explaining how gates can		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				be combined to make a processor or memory.		
3.4.4	Explain Von Neumann architecture. Explain role of main memory, components of CPU, buses. Understand and explain the fetch-execute cycle.	Explain Von Neumann architecture. Explain role of main memory, components of CPU, buses. Understand and explain the fetch-execute cycle.	1 hour	A good way to introduce this is to have old PCs that students can look inside to identify the component parts. This could be done with photographs but having real PCs makes it more interesting. The role of the components needs to be explained. Students only need a high-level understanding of the fetch-execute cycle. They don't need to know the details of register operations etc. A range of online simulators can be used to illustrate this.	Online notes, including some videos (note some content not required): **bbc.co.uk/education/guides/zmb9mp3/revision** Animation of fetch-execute cycle (more detail than needed): **scratch.mit.edu/projects/2145440/** .	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.4.4	Understand difference between main memory and secondary storage and between RAM and ROM. Be able to explain volatile and non-volatile. Explain the effect of clock speed, number of cores and cache size on processor performance.	Understand difference between main memory and secondary storage and between RAM and ROM. Be able to explain volatile and non-volatile. Explain the effect of clock speed, number of cores and cache size on processor performance.	0.5 hours	This is not a very practical topic. Most of the content is probably best explained to the students by the teacher, although students could be asked to research parts of it eg what cache is and how it improves performance. With regard to RAM and ROM, it is helpful to focus on their uses.	Online notes, including some videos (note some content not required): **bbc.co.uk/education/guides/zmb9mp3/revision** Notes on factors affecting processor performance: **bbc.co.uk/education/guides/zws8d2p/revision/2**	
3.4.4	Be aware of why secondary storage is needed and the different types of secondary storage. Explain the	Explain the operation of solid state, optical and magnetic storage. Discuss their relative advantages. Explain what cloud storage is and compare	1.5 hours	It is useful to have physical devices for students to look at here – a disassembled hard disk drive and CD-ROM drive or similar. There is less interest to see inside a solid state drive.	Disassembled storage devices, websites such as **howstuffworks.com** Storage device summary: **bbc.co.uk/education/guides/zxgkxnb/revi**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	operation of solid state, optical and magnetic storage. Discuss their relative advantages. Explain what cloud storage is and compare it to local storage.	it to local storage.		There are also lots of animations available on the internet on website such as **howstuffworks.com** which illustrate the principles behind the operation of these devices. Students could make a presentation to explain how each device works. The relative advantages of the devices should be considered in relation to criteria such as maximum capacity, cost per megabyte, robustness, power consumption and portability. Many students will be familiar with using cloud storage such as OneDrive or Apple or Google's cloud storage systems so this	**sion/6** Explanations of how devices work: **howstuffworks.com/hard-disk.htm**, **electronics.howstuffworks.com/cd.htm**, **computer.howstuffworks.com/solid-state-drive.htm** The cloud: **computer.howstuffworks.com/cloud-hard-disk.htm**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				aspect of the specification would work well as a discussion with students explaining what they use it for and considering the practical benefits that they have seen themselves, but also the risks.		
3.4.4	Understand the term "embedded system" and explain how an embedded system differs from a non-embedded system.	Explain what an embedded system is and how an embedded system differs from a non-embedded system. Give examples of embedded systems.	0.5 hours	This is a relatively small topic. Students need to understand that many computer systems are embedded in other devices and the constraints and differences that this produces when compared with non-embedded systems. Students could be given some scenarios (eg washing machine) and be asked to consider what functionality the system would need and why a non-embedded system		

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				would not be suitable. Differences such as processor speed, amount and type of main memory, secondary storage, input and output devices and upgradeability could be considered.		
3.4.1, 3.4.3	Define the terms hardware and software and understand the relationship between them. Explain what is meant by systems software and application software and be able to give examples of them. Understand the need for and functions of the	Define the terms hardware and software and understand the relationship between them. Explain what is meant by systems software and application software and be able to give examples of them. Understand the need for and functions of the OS and utility programs.	0.5 hours	This topic is very much a theory topic so probably best delivered by the teacher talking and discussing with the class. For the first point, students simply need to know that hardware is that the electronic or electro-mechanical components of the computer and that software are the programs that run on the hardware and tell it what to do to perform a task. Students need to know that application software	Operating systems online notes: **bbc.co.uk/education/guides/ztcdtfr/revision/1**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	OS and utility programs.			completes user-oriented tasks that the user would need to do with or without a computer whereas system software performs tasks related to the management of the computer system. Students need to know that the OS manages processor(s), memory, I/O devices, applications and security but do not need to know how. A utility is a program that helps manage a computer but is not core to its operation eg a compression program, a virus-checker. It might be useful to make students aware that increasingly utilities are being bundled with the OS.		

3.5 Computer networks

Total time for teaching this section: 4 hours.

Many of the topics in this section are quite theoretical, and so could be delivered through discussion and students using textbook/notes. It is important that students have the opportunity to demonstrate their understanding by answering questions. Some practical networking could be done using, for example, Raspberry Pi computers which students can build a network from themselves, but this is not a requirement.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.5	Define what a computer network is. Discuss the benefits and risks of computer networks. Understand that networks can be wired or wireless. Discuss the benefits and risks of wireless networks as opposed to wired networks.	Students should be able to explain what a computer network is, discuss risks and benefits of networks and the relative merits of wired and wireless networking.	1 hour	Students will have direct experience of using networks, both wired and wireless, so this makes a good discussion topics – pros and cons of having a network and also of wired vs wireless networks. Devices such as Raspberry Pis could be used to build a network if it is desired that students have some practical experience.	Online notes and test (note also covers some topics that are not required): **bbc.co.uk/education/guides/zh4whyc/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.5	Describe the LAN and WAN types of computer network. Explain the following star and bus physical network topologies.	Students can describe LAN and WAN and understand star and bus topologies, including their relative merits.	0.5 hours	Differences between LAN and WAN should be considered in terms of size, ownership and the hardware used. Topologies are best visualised; it is worth noting that physical bus networks have limited applications nowadays. This topic can be taught as a discussion or there are many online videos and resources.	Online notes and test (note also covers some topics that are not required): **bbc.co.uk/education/guides/zh4whyc/revision** Video (covers more topologies than needed): **youtube.com/watch?v=WwEZR2vU1UA**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.5	Define the term 'protocol'. Explain the purpose and use of common network protocols including: Ethernet, Wi-Fi, TCP, UDP, IP, HTTP, HTTPS, FTP, SMTP, IMAP.	Students understand and can describe what a protocol is. Students can explain the purpose of the protocols and their use. They don't need to know any technical details about implementation.	1 hour	This topic is very theoretical and is probably best taught with students reading notes or the teacher delivering a presentation. Students should then answer questions that test their understanding. It is possible to demonstrate the use of some of the protocols, for example by using Telnet to open connections to a web server or email server, but this is not required for GCSE. Some online resources are also available.	Online notes (pages 5 and 6 have basic coverage of protocols) **bbc.co.uk/education/guides/zh4whyc/revision**.Too much content for students, but a useful reference for teachers: **wikipedia.org/wiki/Internet_protocol_suite**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.5	Understand the need for, and importance of, network security. Explain the following methods of network security: authentication, encryption, firewall, MAC address filtering.	Students understand why security is important on networks (more so than standalone computers) and the listed security measures.	0.5 hours	This topic can be taught theoretically or, if the teacher has access to this, students could be shown how some of these measures are used in school, for example firewall rules used.	Online notes and test: **bbc.co.uk/education/guides/zs87sbk/revision** Video showing use of MAC address whitelist: **youtube.com/watch?v=SMf-2LjikPA** Very short video on firewalls: **youtube.com/watch?v=wf2ikTtz_gk**	
3.5	Describe the 4 layer TCP/IP model. Understand that the HTTP, HTTPS, SMTP, IMAP and FTP protocols operate at the application	Students should know what the four layers are and some functions of each layer together with which of the protocols listed operate at which layer.	1 hour	This topic is a fairly theoretical one. Students could use textbooks, online notes or videos to learn from. Students need to understand why a stack is used (abstraction), what the 4 layers are and some	Video: **youtube.com/watch?v=mRd79VFkSvQ**	The specification uses the 4 layer TCP/IP model. There is an alternative 5 layer model and also a 7-layer OSI model, which are not required.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	layer. Understand that the TCP and UDP protocols operate at the transport layer. Understand that the IP protocol operates at the network layer.			functions of each layer of the stack and at which layers the listed protocols work.		

3.6 Cyber security

Total time for teaching this section: 2.5 hours.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.6, 3.6.1, 3.6.1.1, 3.6.1.2	Define the term cyber security and be able to describe the main purposes of cyber security. Understand and be able to explain the following cyber security threats: social engineering techniques, malicious code, weak and default passwords, misconfigured access rights, removable media, unpatched and/or outdated software. Describe what	Be able to explain cyber security and the cyber security threats covered by the specification.	1.5 hours	This topic works well as a class discussion as most students will be familiar with some of these topics from their own personal experiences. Students could make a presentation, each focusing on one or more topics.	Lots of cyber security resources including lesson plans and games: **bigambition.co.uk/secure-futures/** Documentary on cyber crime in the UK: **youtube.com/watch?v=8NAbV6w-KUQ** Five of the worst computer viruses: **youtube.com/watch?v=DF8Ka8Jh0BQ** Notes on some topics of computer security: **bbc.co.uk/education/guides/zs87sbk/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	social engineering is. Explain the following forms of social engineering: blagging, phishing, pharming, shouldering Define the term 'malware'. Describe the following forms of malware: computer virus, Trojan, spyware, adware.					
3.6.1, 3.6.1.1, 3.6.1.2, 3.6.2	Describe how social engineering can be protected against. Describe how	Be able to describe methods that are suitable for protecting from cyber security threats.	1 hour	This topic works well as a discussion as students will be aware of some of these topics from their own experiences. They may need to be focused somewhat to ensure that	Lots of cyber security resources including lesson plans and games: **bigambition.co.uk/secure-futures/** Novalabs cyber	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	malware can be protected against. Understand and be able to explain the following security measures biometric measures, password systems, CAPTCHA, using email confirmations, automatic software updates. Explain what penetration testing is and what it is used for.			they cover all of the topics on the specification. A range of useful online videos are available.	security protection game: **pbs.org/wgbh/nova/labs/lab/cyber/** Cyber security threats and solutions: **youtube.com/watch?v=fyh05k83js8**	

3.7 Databases

Total time for teaching this section: 5.5 hours.

Note that the duration of some of the activities will depend upon students' prior exposure to databases and also the number of exemplar activities that they are given to complete. As this is a practical topic, most of the time should be spent by students building and querying databases.

After covering this topic, able students may wish to look at how they can use SQL from within their own programs to manipulate data in a database, but this is not a requirement.

Teachers who wish to develop further their own understanding of relational databases and SQL may wish to read the chapter on relational databases in the textbook "A Level Computer Science AQA Unit 2", published by ECS Ltd (**educational-computing.co.uk/**).

Specification reference	**Summary of the specification content**	**Learning outcomes** **What most students should be able to do**	**Suggested timing (lessons)**	**Possible teaching and learning activities**	**Resource**	**Examination 'hints and tips'** **Students should:**
3.7.1, 3.7.2	Explain the concept of a database. Understand the following important database concepts: table, record, field, primary key. Be able to choose appropriate data types and (where appropriate) lengths for fields	Understand the appropriate database concepts and be able to create a database table. Note that students only need to be able to create tables using a visual table editor. They don't need to use SQL to do this.	1 hour	Students should be shown an existing single-table database and have the opportunity to look at the structure of this. They should then have the opportunity to design and build single-table databases for one or two other scenarios. They can type some data into these databases to check data types etc are appropriate, but there is little purpose to students spending a lot of time typing data in.	Students need a single-table database to look at the structure of before creating their own database(s). Online notes and test: **bbc.co.uk/education/guides/zfd2fg8/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	in a table. Be able to select a suitable primary key for a table.					
3.7.3	Be able to use SQL to retrieve data from a single table database, using the commands: SELECT, FROM, WHERE. Be able to use SQL to insert data into a database using INSERT INTO. Be able to use the SQL commands UPDATE and DELETE FROM to edit and delete data in a	Students can use SQL to query, add data to and update a single-table database.	1.5 hours	If students have no previous experience of querying databases, it may be a good idea to let them do some queries by using query-by-example in a package such as Microsoft Access before moving on to making queries using SQL. Students should be shown the basic SQL syntax and some example queries and other commands. They should then be given an existing database to query and modify using SQL to achieve specific outcomes set by the teacher eg list all the cars	Students need access to a single-table database and a list of tasks to complete with it using SQL. Simple examples of SQL for a single-table database: **bbc.co.uk/education/guides/z37tb9q/revision** SQL tutorial: **w3schools.com/sql/**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	database.			that are made by manufacturer A and are colour B, update the value of the car X to be Y.		
3.7.1, 3.7.3	Explain the concept of a relational database. Understand the following important database concepts: foreign key. Understand that the use of a relational database facilitates the elimination of data inconsistency and data redundancy. Be able to use SQL to retrieve	Students should understand what a relational database is, that a relational database eliminates redundancy and inconsistency and that relationships are modelled using primary and foreign keys. They should be able to write queries to retrieve data from a database containing up to three tables.	1.5 hours	Students could be shown a single-table database that contains redundancy and could lead to inconsistency and then a two-table database that eliminates this problem. They should then have the opportunity to query a two-table database using queries that cross-reference both tables. This should finally be built on by students querying a three-table database.	Students will need access to a two-table and a three-table database and a list of questions to answer from the database using queries. A basic introduction to relational database concepts: **bbc.co.uk/education/guides/z37tb9q/revision** SQL tutorial: **w3schools.com/sql/**	Exam questions will not use databases consisting of more than three tables.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	data from a relational database using the commands: SELECT, FROM, WHERE.					

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.7.2	Be able to produce a design for a relational database from a description of a scenario.	Students can specify suitable tables, fieldnames, data types and field lengths and primary and foreign keys for a scenario involving a database containing up to three tables.	1.5 hours	Having had previous experience of querying a relational database students should now be presented with two or three scenarios and be asked to design relational databases for them. It is helpful if the scenarios are ones that students are familiar with, for example, students in a calls or players in a football team in a league. This can be done as a theoretical exercise but students would likely benefit from actually building at least one of the databases that they have designed and using it to perform some queries.	Students need scenarios to design databases for. A basic introduction to relational database concepts: **bbc.co.uk/education/guides/z37tb9q/revision**	Exam questions will not require the use of more than three tables.

3.8 Web page design

Total time for teaching this section: 4.25 hours.

This is a practical topic; most of the time should be spent by students gaining practical experience of building websites.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
3.8.1	Know that web pages can be constructed using HTML and CSS. Know that HTML is used to create the structure of a web page and that CSS can be used to change the style of the page. Understand that server-side scripting languages can be used on a web server for dynamic generation of web	Students should have an understanding of what the basic web technologies are and how they fit together, including HTML, CSS and server- and client-side scripts.	0.5 hours	Students should be given an overview of the key web technologies, with an emphasis on HTML and CSS which are the languages that students will use in subsequent lessons. However, they should also be made aware of how scripting can also be used. The suggested website contains a video explaining how this is used on the BBC website.	Introduction to web development concepts (goes beyond spec in places): **bbc.co.uk/education/guides/znkqn39/revision**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	pages. Understand that client-side scripting languages can be used to add additional functionality to web pages.					
3.8.2	Be able to create the structure of a web page using HTML. General: <html> </html>, <title> </title>, <body> </body> Block-level: <p> </p>, <h#> </h#>, <ol> </ol>, <ul> </ul>, <li> </li> Inline: <strong> </strong>, <em>	Students should be able to understand and write HTML code using the tags covered in the specification.	1.5 hours	Students should look at the structure and code of simple web pages that use the HTML tags listed. They should be given the opportunity to build a small website consisting of a couple of pages including images and hyperlinks that only uses HTML. The only software required is a text editor and a web browser. A syntax-aware text editor such as	HTML editor and preview: **w3schools.com/html/tryit.asp?filename=tryhtml_default** HTML tutorial: **w3schools.com/html/default.asp**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	</em>, <br />, <a> </a>, <img />			Notepad++ will help make the structure of code clearer. Commercial tools such as Adobe Dreamweaver could also be used. HTML code can also be written and tested on the w3schools website.		
3.8.2	Be able to use basic CSS to control the layout of a web page. HTML tags: <style> </style>, <div> </div>, <span> </span> A style rule is made up of three parts, a selector, a property and a value: selector {property : value}	Students should be able to understand and write CSS rules to format a web page using the selectors set out in the specification.	2 hours	Students should be introduced to the concept of how CSS can be used to change how a web page is formatted. They should have the opportunity to build a web site (or modify a previously build website) using the CSS selectors required by the specification. There is the possibility of students working in groups on this topic to	Website that illustrates the power of CSS: **csszengarden.com/** CSS tutorial: **w3schools.com/css/default.asp** CSS editor and preview: **w3schools.com/css/tryit.asp?filename=trycss_default**	For the exam, only embedded style sheets will be covered but students might wish to use external style sheets for practical work.

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
	Two kinds of selector need to be known: type, class. The following CSS properties should be known: background-color, color, text-align, font-family, font-size, font-weight, font-style.			build a website with more content.		
3.8.2	Be aware of the key differences between HTML5 and earlier versions of HTML.	Students should be aware of the key differences between HTML5 and earlier versions of HTML.	0.25 hours	Students should know that HTML5 supports many facilities that were previously only available by using add-ons to HTML, supports provision for the development of web applications through support for scripting and access to many new APIs, supports the ability to more easily incorporate graphics and multimedia elements, and employs	Summary of what is new in HTML5: **techradar.com/news/internet/web/html5-what-is-it-1047393** Some demos of what HTML5 can do: **html5demos.com/**	

Specification reference	Summary of the specification content	Learning outcomes What most students should be able to do	Suggested timing (lessons)	Possible teaching and learning activities	Resource	Examination 'hints and tips' Students should:
				the use of new semantic elements such as <header> and <article> to define different parts of a web page.		

Assessment and exam preparation

Total time for teaching this section: 9.25 hours.

Throughout the course, it is important that students are assessed formally, both on their programming skills and on their knowledge of the more theoretical side of the course.

Completing programming tasks under timed conditions and without teacher assistance will help prepare students for the on-screen programming exam at the end of the course. Also, the teacher can use the results of such assessments to identify students who are finding the work challenging so that they can intervene early to help them.

The skeleton program for the on-screen programming exam will be released three months before the date of exam in the year of the exam. Students should spend time working with this program and improving it so that they are prepared for the exam.

A range of assessment resources, including example question papers and mark schemes are available from Oxford AQA Exams.

Assessment materials prepared for English GCSE exams would also be suitable to use for exam preparation. Example materials for the AQA GCSE is available at: **aqa.org.uk/subjects/computer-science-and-it/gcse/computer-science-8520/assessment-resources**.

GET HELP AND SUPPORT

Visit our website for information, guidance, support and resources at **oxfordaqaexams.org.uk**

You can contact the computer science subject team directly;

E: **computerscience@oxfordaqaexams.org.uk**

OXFORD

INTERNATIONAL
AQA EXAMINATIONS

OXFORD INTERNATIONAL AQA EXAMINATIONS
LINACRE HOUSE, JORDAN HILL, OXFORD, OX2 8TA
UNITED KINGDOM
enquiries@oxfordaqaexams.org.uk
oxfordaqaexams.org.uk