GCSE Computer Science

Example Scheme of Work

AQA GCSE Computer Science
Example Scheme of Work (120 guided learning hours)

Introduction

This is based on the new GCSE Computer Science delivered over 120 guided learning hours. This includes the 50 hours for both controlled assessments which are not included in the scheme of work.

This scheme of work is provided as an example only. Schools/colleges should produce their own assessments in line with their local delivery circumstances. It should be noted that this scheme of work assumes that students have had no previous experience of formally taught computer science.

This scheme of work includes reference to various development tools and languages and, where appropriate, specific examples have been provided. Please note that the only mandatory language required by the specification is SQL. All other languages and development environments are the choice of the school/college and/or individual student.

The specification is intended to be engaging and relevant and the majority of the theory exists to increase the students' ability to design and create programmatic solutions to computational problems. This scheme of work aims to avoid the somewhat artificial distinction of content between practical and theory. There are three suggested extended homework projects within the scheme of work. These are not necessary to cover the specification but they may be useful as programming practice and also as preparation for the controlled assessments.

Some parts of the course are not explicitly covered in lessons (such as pseudocode and commenting programs) as it is intended that they will be used and embedded in work throughout the course.

Useful resources

You may find these websites valuable for further resources (this is not an exhaustive list):

- BYOB (Build Your Own Blocks), a visual programming language primarily for children,
  http://byob.berkeley.edu/
- Codecademy, a web-based learning environment currently supporting JavaScript and web programming, Ruby and Python,
  http://www.codecademy.com/
- Computer Science For Fun, shows how computer science is also about people, solving puzzles, creativity, changing the future and having fun,
  http://www.cs4fn.org/
- Computer Science Inside, provides web based resources for teachers,
  http://csi.dcs.gla.ac.uk/
- Computing At School, joining this group is free of charge and provides access to resources and relevant debate,
  http://www.computingatschool.org.uk/
- CSUnplugged, a collection of free learning activities that teach Computer Science through engaging games and puzzles,
  http://csunplugged.org/
- Lazurus, a software application that provides a Free Pascal compiler,
  http://lazarus.freepascal.org/
- Microsoft DreamSpark, gives free access to professional-level development, design and gaming software to build real sites, apps and games for XBoxLive, Kinect and mobile phones,
  http://www.dreamspark.com/

- Microsoft .NET Gadgeteer, a paid-for platform that allows hobbyists, educators and developers to build and refine prototype electronic devices, http://www.netmf.com/gadgeteer/

- MIT AppInventor, use a browser to design then, through a live connection between the computer and a phone, the app appears on the phone. http://www.appinventor.mit.edu/

- MySQL, allows the user to create of a relational database structure on a web-server in order to store data or automate procedures, http://www.mysql.com/

- PHP, a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML, http://www.php.net/

- Raspberry Pi, a credit-card sized computer that plugs into your TV and a keyboard and is for children to learn programming through practical activitiy,
  http://www.raspberrypi.org/

- Scratch, a programming language that makes it easy to create interactive stories, animations, games, music, and art, http://scratch.mit.edu/

- W3Schools, a reference and tutorial for HTML, CSS and JavaScript and other languages for the web, http://www.w3schools.com/

- Young Rewired State, offers support to under 18s who are learning how to program, http://youngrewiredstate.org/

Topic	Spec	Hour	Activities	Resources	Comments
Computer Systems	3.1.8.1	1st	• Demonstrate a modern computer game. Ask students to decompose what they see going on into sections – input/output mechanisms, AI, graphics, sound, information to be saved and so on. • Focus on the controller and break down in some detail the passage of data through the system. • Explain that a computer system is a combination of hardware and software working together.	• Any games system will suffice but use of the XBOX (or PC with XBOX controller) and Kodu gives an additional benefit that you can show basic sequencing of instructions and response to events. • If the resources are available you could use a games system with a transparent case to illustrate the flow of data.	• Ensure that students realise a computer system does not have to look like a games box or a PC and can be any combination of hardware and software.
Expressions & Types	3.1.1	1st	• Use a command line interpreter as a calculator introducing the main arithmetic operators (*, /, +, - and power) and the need for parentheses in numerical expressions. If an interpreter doesn't exist for your choice of language then introduce these expressions in a simple program. • Show students how a function of their language can be used to inspect the type of a given expression (for instance `type(3)` might return `integer`). Students to experiment with the types of numerical expressions. • Explain the number types integer and real with respect to the programming language	• Command line interpreters exist for many languages and increasingly many can be accessed as web applications. • See this article for more on type introspection: http://en.wikipedia.org/wiki/Type_introspection	

			used. • Extend the expressions to include comparative operators ((>, <, ≤,≥,≠) and ask students to use the same introspection function from earlier in the lesson to introduce the Boolean type. • Homework could be to extend these expressions and introduce the types string and character.		
Variables & Constants	3.1.1	1st	• Introduce the concept of variables using labelled plastic cups and tokens. The value of a cup is the number of tokens it contains at any given time. • Introduce the pseudocode for assignment and set exercises for students involving the cups. • Show students how to declare variables and assign values using an interpreter for their language (again, if an interpreter doesn't exist then frame these in a simple program). • Extend questions to include values that do not change throughout the computation and introduce the idea of a constant as an "unchanging variable". • Set homework to answer questions abstracted away from the language such as, "swap the contents of variables A and B".	• The cups and tokens activity works better if the students are also given sticky labels that can be used to label the cups. By stepping through the assignments in sequence you can also show how the state of variables changes throughout execution of a program. • The AQA pseudocode can be found at: http://web.aqa.org.uk/qual/newgcses/ict/computer-science-materials.php?id=10	• Part 3.1.1 of the specification states that students should understand when to use constants and variables in problem solving scenarios. Although this lesson starts this it is important that the first lessons in programming reinforce the correct use of the fundamental techniques such as the use of variables and constants and sequence, selection and iteration even when they are not explicitly covered.
Selection	3.1.3	1st	• Recap on the Boolean type from the earlier lesson. • As a class create a simple number guessing game and record the result in pseudocode. Hopefully this will intuitively introduce IF	• Refer to the AQA document on the use of pseudocode – this isn't essential but it will be the pseudocode used in	

		statements. • Convert the pseudocode into a program and extend the program to include basic user input and output messages. • Extend the game to include IF-THEN-ELSE statements first in pseudocode and then in the program.	the exam and so early familiarity would benefit students. • If not already done this would be a good time to introduce students to how to create and execute a program in your preferred development environment.	
	2nd	• Students to arrange themselves randomly in a line and then perform a parallel pairwise sort based on the outcome of 'are you taller than me?' The same could be completed on any Boolean condition such as 'are you older than me?' • Get students to think of a number between 1 and 64 and then try out in pairs how long it takes them to guess the number asking only Boolean question (i.e. where the answer can either be true or false). If done perfectly this should take a maximum of 6 questions. • Simplify the binary search to numbers 1 to 4 and show how to find the answer using a binary tree. Use this tree to introduce the idea of nested conditionals.	• CSUnplugged have a fascinating implementation of a parallel sorting exercise that can be done on a playground with some chalk and sunny weather: http://csunplugged.org/sorting-networks • A tree diagram (that also shows the link with binary numbers) for 0-15 can be found here: http://ecee.colorado.edu/~mathys/ecen1200/hwcl06/bin16tree2_640.png	• The idea behind the parallel sort is to show that computation involves asking simple logical questions in sequence. Also that computers cannot infer answers outside of their instructions, for instance they can't think 'I'm the tallest person here so I'm going to walk to the end'.
	3rd	• Show students how to code using nested conditionals using the above binary search as an example.		• This lesson can also be used to introduce more advanced features of the

	• Discuss the problems with deeply nested conditionals when developing solutions. • Set homework to extend the game to guessing numbers between 1 and 8 and produce a coded solution.	development environment used.

Repetition	3.1.3	1st	• Exhibit a program that uses audio playback features of a language to play different musical snippets such as a drum beat. • Show how these can be played multiple times using repetition statements such as REPEAT 10, FOR X IN 10 or similar. • Students to create their own music using repetition. • Incorporate IF statements to respond to user input and adjust volume level, instrumentation or similar.	• There is an abundance of free and copyright-free music that can be sourced from the web and could be used for these exercises.	• Different languages vary in the ease of playing different sounds; Scratch for instance makes this trivially easy, others might require OS-dependent libraries.

	2nd	• Revisit the binary search from earlier in the course for the range 1-8 and, having provided students with a tree diagram showing the choices to be made, get students to encode a solution in pairs using repetition. • Discuss the answers that students get and offer an exemplar. Question whether this has made the solution more correct, more efficient or more elegant.	• Giving students a tree diagram showing the comparison choices made at each step will enable students to focus on the mechanics of using repetition. A tree diagram (that also shows the link with binary numbers) for 0-15 can be found here: http://ecee.colorado.edu/~mathys/ecen1200/hwcl06/bin16tree2_640.png	• Students will hopefully produce code that is easier to follow than earlier in the course when answering the same question without repetition. Although this may not be more efficient it will make the code easier to read, easier to update and possibly easier to avoid making mistakes.
	3rd	• Prior to the lesson create a simple game environment where 'sprites' respond to user input. • Students to adjust sprite behaviour using WHILE statements. This could be movement speed, interaction with other sprites and so on such as WHILE right button pressed TURN 1 degree clockwise.	• There are plenty of environments that allow students to write games without too much additional complexity (Greenfoot, Scratch, YouSrc, PyGame and so on).	
	4th	• Students to create two pseudocode programs to find out if any of the unsorted group of students has a birthday today, one using REPEAT and one using WHILE. Discuss the run-time efficiency of both of these approaches (the WHILE approach has the advantage that on average it will require less inspection of birthdays).	• Students should realise the difference in uses of count controlled loops such as REPEAT and condition controlled loops such as WHILE. • The AQA GCSE Computer Science pseudocode	

			• Students to create pseudocode for various cooking recipes such as baking a cake or making ice cream by using variables, constants, sequence, selection and repetition. • Set homework to create programmatic solutions problems of varying complexity using a combination of selection and repetition.	guide shows what loops may be referred to in the exam although students should be familiar with both these and all of the looping mechanisms that will enable them to use their programming language expressively.	
Boolean Operators	3.1.1	1st	• Play a guessing game where you have to narrow down people based on their appearance to introduce NOT, AND and OR. • Use truth tables to build on the intuitive notions of these operators. • Set exercises to simplify Boolean expressions of increasing complexity using truth tables.	• There are quite a few online guessing games (and also the proprietary Guess Who) that will work with this - a web search will return plenty.	• This should be fairly intuitive although the case of OR where it is either or both may have to be covered more clearly. • Simplification of Boolean expressions is not on the specification although it is included here to help understanding or as an extension activity.
		2nd	• Show students how nested conditionals can sometimes be better replaced with Boolean statements combined with AND or OR. • Students to use a gaming environment where sprites respond differently depending on, for instance, whether the x and y coordinates are within certain bounds.	• Reuse the gaming environment from earlier in the course. Many of these come with scenarios and sample programs that can be adapted and enhanced.	• This will give additional help to students who are going to complete the controlled assessment in gaming.
Arrays	3.1.1	1st	• Motivate the use of arrays by showing data sets (such as names of everyone in the class)	• Depending on the language and whether	• If the exam refers to arrays it will be assumed

		represented as arrays as opposed to using individual variables. • Show how to declare arrays and assign elements. This will obviously vary depending on the programming language used. • Students should create arrays for simple data sets such as days of the week, players in a team and prime numbers.	the length of an array can be changed after it has been created it may help to illustrate the use of arrays with a fixed series of indexed boxes.	that indexing starts at 1 and not 0. Although this will not affect the way students program in their given language they should be aware of any difference as part of their exam preparation.
	2nd	• Introduce the techniques and idioms used to iterate over arrays in the given language, eg FOREACH or FOR constructs. • Show how to use random number generation to populate an array and use iterative techniques to print out all of the numbers, find the highest and lowest number, the mean average, the most common number and so on. • Set students homework to analyse a large array populated with words found in a book/play/newspaper article or similar. Students could find out the average word length, the relative occurrence of popular words, the occurrence of different characters and so on.		• This is potentially a very language dependent part of the course so relate the programs used to pseudocode, animations and so on in order for students to see the general picture as well as the language techniques.
	3rd	• Show how arrays can contain other arrays as elements by creating a simple noughts and crosses game. • Students to create an array that represents a paragraph where each element is a sentence. Each sentence is in turn an array of words. They could then use nested FOR loops (or	• There are various physical ways to show this such as envelopes containing envelopes containing cards.	• Depending on the students understanding they may be ready to develop the noughts and crosses game further as homework. This would be particularly useful

			similar) to print out the group of sentences in full, print individual sentences backwards, print the whole paragraph backwards, reassign some words and so on.		before introducing functions to further motivate that topic.
Flowcharts	3.1.3	1st	• Ask students to create a sketch or picture of a program that represents the structure without relying heavily on text. Gather answers and pick up on good unambiguous pictures. Some students will probably use arrows to show flow of execution. • Show the standard flowchart symbols for start/stop, process, I/O and decision. • Students complete exercises in translating flowcharts to programs and vice versa. Iteration should be included and the link between feedback on a decision in a flowchart and a WHILE loop should be explored. • Discuss the advantages and limitations of flowcharts when developing and analysing programs.	• Flowol is commonly used to introduce flowcharts, especially as students can visualise the results.	• Flowcharts may well have been covered in Design and Technology courses or ICT in KS3. If so the focus on this lesson should be on conversion between the programming language and flowcharts.
Trace Tables	3.1.6	1st	• Present students with complex pseudocode and flowchart algorithms and discuss how their meaning could be found out. • Return to the second lesson on variables and the use of labelled cups and use these to trace through the algorithm. • Repeat this process but tabulate the results as execution progresses, present the result as a trace table. • Return to the algorithms presented at the	• The introduction of trace tables is best motivated with algorithms that appear complex but present an obvious answer once they are worked through. Many sorting algorithms are in this category.	• If there is time (and if it is appropriate for the development environment used) students could be shown how their IDE can keep track of variables and how they can be inspected by pausing execution at breakpoints.

			start of the lesson so students can use trace tables to ascertain their meaning.		
Errors	3.1.6	1st	• Discuss problems that students have had so far with the development of their programs and categorise the errors (leaving aside lack of knowledge) as syntactic, logical and runtime. • Show how their IDE can be used to help with syntax errors using code highlighting, automatic indenting, code completion, code suggestions or similar. • Make a list of the most common syntax errors in the language used and display them in the classroom. • Present the students with flowcharts and pseudocode of algorithms that contain logical errors. Explain what the algorithms are intended to do and ask students to correct them. These algorithms could also be translated into programs for further practice.		• At this stage students will probably be most concerned with syntax errors although they will probably have encountered logical errors too even if they didn’t realise it. Runtime errors will probably be less common but may have occurred when covering arrays (indexing errors).
		2nd	• Continue the work on errors by providing pseudocode and flowchart algorithms that are known to cause runtime errors in the chosen programming language. • Students should encode these algorithms and run them and inspect the output when the program crashes. Students should correct these programs so they run correctly. • Finally show students how errors can be ‘caught’ and handled in their programming language. Discuss the reasons for doing this and also the possible dangers.	• This is another section that is highly dependent on the language chosen although runtime errors are commonly generated by type mismatches (such as inputting a string when a number was expected) and array indexing errors (such as using -1 as an index). A web search for common	

				runtime errors in the specified language should provide some pointers(!).	
Functions & Procedures	3.1.4, 3.1.3	1st	• Create a list of the built in functions that students have already used. These may include type inspection, user input and output, array length, random number generation and so on. Explain that as language users these can all be seen as a black box that takes inputs and returns an output. • Take the example of working out array length and ask students to write the program for how the inner workings of the function could be written. • Show how this code can be 'wrapped up' in a function in the language used with the array used as an argument and an integer returned. • Repeat these steps for a function to return the bigger of two numbers but let the students encode the function. • If time, repeat again for a function to return whether a string contains an odd number of letters.		• Different languages treat functions in different ways (such as the way in which parameters are handled, whether the type of the return value is enforced, whether parameters have to be arranged in a certain order and if they are allowed default values and so on); students should be able to use functions capably within their chosen language.
		2nd	• Ensure students are familiar with the terminology parameters and return value. This can be illustrated with two or three differently coloured paints poured into a jar that, after shaking, returns a different coloured paint. It can also be shown programmatically by assigning the result of a		

		function to a variable. • Use type introspection to show that the type of a function applied to its arguments is the type of the return value. • Show students how to access the documentation of the inbuilt functions of the language and set tasks that require reading the documentation.		
	3rd	• Build on the previous lesson with tasks to build functions of differing complexity. These functions should involve manipulating numbers, text, arrays and so on. • Introduce procedures as blocks of code that can take input arguments but do not return a value.	• Asking students to rewrite some common built in functions (such as MAX, LENGTH, INDEXOF and so on) will give the students an easy way to test their answers.	
	4th	• Formalise the concept of testing; students will already have tested their programs to check they run as expected but their testing may have been ad hoc and imprecise. • Show how unit tests can be written for functions to check they run as expected. • Students should return to the functions they have written previously and include unit tests for them.		• This is another area where the language used will dictate the best way to deliver this concept. If a language does not support unit testing directly the students could be shown how to develop test programs that run the functions and compare expected outcomes with the actual results. • Throughout the creation

					of unit tests students should be encouraged to comment their work in a consistent and sensible way.
		5th	• Show a program that contains three or four correctly written functions. The program could be on any topic - a numerical problem, a game, a program to display the output of a webcam – as long as it is clearly separated into functions. • Provide students with the code for a selection of functions all related to a problem and challenge them to solve problems by application of these functions. For example you could have a function that returns the average combined weight of any number of passengers, a function that returns how many litres of fuel a car uses per kilometre given the weight of passengers, a function that returns the price of a given number of litres of fuel and a function that returns the distance between two known points. Students should then work out the price of journey scenarios. • Students may make unnecessary use of variables to hold values returned from function calls. Show how function calls can be embedded within expressions to reduce syntactic clutter.		
Scope	3.1.5	1st	• Give students code to experiment with where (legitimate) name clashes occur such as the		• For the purposes of the exam students will not be

			same use of name for the parameter and the argument. Allow students to formulate their own rules on how their programming language scope works. • Colour coding variable names on a large print out of code and pinning string from name occurrences in code to a table where their values are kept may help explain the concept of scope.		required to explain scoping rules for any specific language although this is desirable for their own development and the controlled assessment if they are to use it to program effectively.
Bespoke Data Structures	3.1.2	1st	• Students to create a simple bespoke data structure that contains details for a person at school (eg firstname, lastname, age and gender). • Students should be shown how to create multiple instances of this data structure and assign them to variables. • Students should be comfortable with accessing and updating the fields/attributes of the data structure.		• It may be easiest with some languages to touch on object orientation at this point if this is a strength of that language, others may wish to discuss this in terms of records or other aggregate structures. There is no need to cover OOP in the specification but it may be beneficial if you intend using environments for the controlled assessments that rely on object orientation.
		2nd	• Continue work on bespoke data structures by getting students to model geometric shapes (could use POINT with two floating point numbers for x and y coordinates as a starting point and then for RECTANGLE include a		

			POINT for the bottom left corner and two other floats for the length of the sides, alternatively a circle could be a POINT for the origin and a float for the radius and so on).		
		3rd	• Introduce a gaming environment that makes use of bespoke structures. • Create the shell of a game but allow students to use, update and customise their bespoke data structures.	• Greenfoot and PyGame work equally well for Java and Python respectively. BYOB Scratch also has this functionality in a more limited form.	
		4th	• Continue working in the gaming environment. In addition to supporting learning of bespoke data structures this could serve as an introduction to bespoke data structures this could also be preparation for the gaming controlled assessment.		
Software Development Lifecycle	3.1.11	1st	• Reflect on how students have created programs so far – what patterns have people noticed? What has worked successfully and what less so? • Invite a professional developer to discuss how programs are developed and maintained in their sector of the software industry.	• Contacting CaS may help in locating a willing and suitable developer: http://www.computingatschool.org.uk/ http://www.microsoft.com/uk/education/teachers/innovative-teachers.aspx	
		2nd	• Divide students into groups and the teacher takes on the role of different clients. Each group is given tubs of coloured play dough. The 'clients' each have a picture of a structure	• Provide reference to the exemplar test plans for the A-grade controlled assessment. This shows	• For the specification students should know the major stages of development and how

		that they need building – the waterfall clients must describe this structure completely to their group before the group can make it whereas the cyclical team can ask for regular feedback from their client. On completion of the structures ask each group to feedback on the strengths/weaknesses of their approach. • Formalise different approaches to developing software (cyclical, waterfall, spiral). Discuss which approach would work best for students during their controlled assessment. • Show how to create a test plan and explain the different ways that testing can be carried out during software development. Relate this to earlier work on unit testing for functions.	a blend of unit testing and also end user testing.	they fit into the different development models.
Prototyping & Testing 1	*3.1.11.1, 3.1.12* *hw*	• *Students should now be in a position to develop a small scale but non-trivial project. This could be set as an extended homework over two weeks with peer-review sessions held over ten minutes at the start of each lesson.* • *Projects could be as simple as the 'here is some data, do something interesting with it' approach or the project could be more formally structured.* • *Encourage students to adopt an iterative prototype-test-refine approach when developing their project.*	• *This could be left as an open ended task enabling the students the freedom to explore beyond what has been formally taught in the course. An interesting project may be to organise a collaborative online code project –this will require tools that enable sharing/versioning of files such as*	• *Prototyping and testing have already been covered in the lessons, this project will give students the opportunity to use these skills and improve their knowledge of their language.*

				PythonAnywhere or Sharepoint. Alternatively students could be left to develop their solutions individually.	
Bits & Bytes	3.1.10	1st	• Students challenged in groups to work out ways to communicate (generally) with one another using 8 torches. • Explain that the on-off state of a torch is analogous to the binary alphabet used for internal communication within a computer. • Introduce the terminology bit, nibble, byte, kilobyte, megabyte, gigabyte and terabyte.	• Referencing the terms megabyte, gigabyte and terabyte to known items of computer hardware may make the numbers more tangible.	
Number Systems	3.1.10	1st	• Use place-holder cards to create binary numbers. • Introduce and practise binary to denary and denary to binary conversion. • Reinforce use of binary-denary conversion with the Cisco binary game.	• There are lots of well documented ways to teach binary numbers. This approach is from CSUnplugged (see an earlier lesson for links to their resources). Creating personal playing cards with the powers of two on them (and a 0 on the counter side) may make it even quicker for students to grasp and will help with assessment. • The Cisco binary game can be found here: http://forums.cisco.com	• The specification covers conversion of between binary, hexadecimal and denary for the denary values 0-255.

			/CertCom/game/binary_game_page.htm	
	2nd	• Discuss the limitations of binary for humans • Explain how number systems are created and present base 10 and base 16 using rising powers of the base and place holders. Explain that there is nothing special about base 2, 10 or 16 except that one is necessary for digital computers and we have ten fingers to count with. • Practise ease of conversion between base 10 and base 16 to base 2, leading to an understanding of why base 16 is commonly used to represent binary numbers instead of base 10. • Practise conversion between binary, denary and hexadecimal.	• Although it is useful for students to understand how to convert directly to hexadecimal values, they may also benefit from knowing that a nibble can be directly converted to a hexadecimal digit.	• Refer to the specimen exam paper for how marks are awarded for number conversion problems.
	3rd	• Challenge students to write a program in pairs that accepts user input of a number in denary and then converts it to base 2 and base 16. This should be attempted without the number format handling of the language, i.e. the students create their own versions of, for example, the functions `den2bin` and `den2hex`. • Explain that as binary is the only alphabet used by computer systems it must be capable of representing every data type represented in the language. Introduce definitions of data and information in the context of a bit pattern with and without meaning respectively (3.1.1).	• Students could create unit tests for these functions using the built in number conversion functions. • Refer to the language documentation and the underlying operating system architecture for how numerical values are stored in the computer (a typical value would be 4 bytes for integers and floating	

			• As homework investigate the memory space needed for real and integer numbers and characters (or a simplification of these if required) used by your programming language.	points and 8 bytes for a double).	
Character Sets	3.1.10	1st	• Ask students to think of all the characters they would like to encode and map each one to a number. • Look for similarities in the encodings (such as consecutive numbering through the alphabet). Examine what characters other than the alphanumeric ones were encoded. Ensure students see white space as characters. • Introduce ASCII and show how it can be programmed directly (possibly using integer values as characters) directly in the language. • Discuss the limitations of using ASCII in modern (global) computing.	• Provide students with an online ASCII http:/www.asciitable.com/ • It may help to illustrate the last point by loading web pages written using a non-Latin alphabet, such as http:/www.welcome2japan.cn/	• Unicode is not on the specification although it might be useful to present it as an alternative encoding when discussing the limitations of ASCII.
Sound	3.1.10	1st	• If possible play an LP record through analogue equipment and represent a small example of this music as a continuous graph (time against frequency). • Discuss ways this music could be digitised. • Students investigate sample rate and bit depth. • Exhibit the same digital music encoding at different sample rates and bit depths.	• Consult the physics department for any available resources. • Simple audio software such as Audacity can be used to encode music at different sample rates and bit depths although the difference may not be perceptible.	
		2nd	• Explore the in-built functions of the programming language for using and manipulating sound.	• This will obviously vary according to the language.	

Bitmap Images	3.1.10	1st	• Before the lesson create paper versions of a bitmap image using a grid, coloured squares of paper and a key linking x and y coordinates with the colour and set groups the task of 'regaining' the image. • Discuss what meta-data needs to be known if the bitmap is to be decoded. • Investigate the effects of resolution and colour depth on the appearance and file size of an image using digital graphics software.	• Block images work best for the bitmapped image task – anything with a lot of detail will likely not be recognisable at the resolution at which the students will be working. • Images can be created by opening an image in an online photo editor (Pixlr for example) and changing the resolution or canvas size.	

Instructions & the CPU	3.1.8.3, 3.1.8.4	1st	• Convert a small program to assembly language (using the students' language if appropriate). C compilers such as GCC work well for this activity. • Students to examine the assembly for common forms and also the lack of programming constructs often used in high level languages. • Describe the one to one mapping of assembly to machine code and reiterate that all data and instructions are all encoded at the lowest level in binary. • Perform group activities mimicking the working of the CPU (students are given basic assembler – ADD, STORE, LOAD – and are assigned jobs as the program counter, memory, IO and the ALU.	• The AQA AS Computing text book provides some examples of a very basic, generic assembly language and the role of registers and the ALU. This course obviously does not go into the detail required for AS level but it is a good source of reference. • The group activity could be arranged with students assigned the various roles and then only being allowed to pass messages with the students they are connected to.	• The intention here isn't for students to have a firm grasp of assembly language but instead to realise that the computer does not execute the high level instruction and must first convert it to a low level form where instructions are executed serially by being fetched from memory and executed by the processor. • The commands ADD, STORE and LOAD are taken from the existing AQA Computing AS unit COMP2.

		2nd	• Following on from the CPU activity, discuss what could speed this process up – lead to clock speed, number of cores and cache and alter the activity appropriately. • Investigate current and legacy CPUs for size [in terms of number of transistors]. 'Discover' Moore's law. • An extension to this topic could be to investigate future limitations of Moore's law and alternative models of computation such as quantum and biological.	
Memory	3.1.8.4	1st	• Boot into the computer's BIOS and explain how before the operating system starts the computer 'boot straps' with a comparatively small number of instructions. These instructions that are necessary to kick start computers are stored on ROM. • If you have old unused hardware available then you could take some old computers apart and run bench marking test on them using differing amounts of RAM.	• It might help to show inside a PC and show where ROM can be found (the difference in size between a ROM chip and RAM gives a weak but visual clue to the difference in memory size. • The bench marking tests could be informal, such as recording the time taken to load four applications simultaneously on a machine with 512MB and then on a machine with 2GB with otherwise equivalent hardware. Check in advance that

				the results are indicative of RAM size.
		2nd	• Refer back to the previous activity on the CPU for the role that RAM plays in executing instructions. • Explain the terms volatile and non-volatile and ensure students realise why volatile RAM cannot be the only memory in a functioning computer system. • Explain the use of virtual memory to overcome insufficient RAM and data/instructions in non-contiguous, or completely separate, data stores. • Students to act out how virtual memory works in groups.	
Secondary Storage	3.1.8.5	1st	• Simulate the function of optical drives by getting students to hold up a sequence of mirrored paper and black paper. With the lights out shine a torch at the paper and reflect it on to the ceiling. Work along the line in time to 'read' the bits from the ceiling. • In a similar fashion to the above, simulate magnetic storage by getting students to arrange a series of small magnets in a random north-south alignment, suspend another light weight magnet by string and pass it over each of the underlying magnet, reading off the bit interpretation as they go. • Alternatively, and possibly in addition to the above, show one of the many online videos showing how optical and magnetic storage	

			media function such as http://www.youtube.com/watch?v=f3BNHhfTsvk	
		2nd	• Investigate solid state media and compare read and write speeds (at a basic level) with typical optical and magnetic media. • The above could be shown by using a program that accesses the drives directly and saves/reads in the same large file, keeping time as it does so. Students could adapt this program to provide their own values for read/write speeds.	• This would require a prewritten program that accesses known drives on the system. The program could use a timer to indicate transfer speeds to and from these devices.
Input & Output Devices	3.1.8.2	1st	• Categorise 20 commonly found IO devices into input, output or both depending on their function. • Investigate the use of novel input and output devices in the entertainment and mobile industries. Include reference to concept devices and discuss how these could fundamentally change our daily relationship with computer systems (including the ubiquitous use of smart phones).	• Students could discuss their own use of games and mobile devices and also access plenty of the concept videos available from companies to discuss the future of our relationship with these systems.
Prototyping & Testing 2	*3.1.11.1, 3.1.12*	*hw*	• *Develop a program that interfaces with non-standard input devices such as the Kinect. This is another mini-project that students should be encouraged to approach using prototyping and continual testing.*	• *The Kinect is an expensive optional extra if the student does not already have one. Alternatives could be webcams using image recognition software or the Picoboards for Scratch (the latter can be*

				adapted for other languages).	
Algorithms	3.1.9	1st	• Give students a set of balancing scales and 10 identically sealed containers each with a different weight. Ask students to arrange the containers in weight order, keeping a record of how many times they use the scales. Students should try and formalise their approach using flowcharts or pseudocode. • Explain that algorithms are terminating solutions to computational problems such as ordering an array or finding an element in a sorted list.	• Old film cartridge containers filled with 1p coins work well for this exercise, they can also be labelled with a code on their underside to reveal their weight.	• Students should realise that they have used and developed many algorithms in this course already.
		2nd	• Ask students to encode their sorting algorithm and run bench mark tests on them using random number generation to give a measure of their efficiency. • Alternatively, provide students with some simple algorithms in pseudocode and flowcharts and ask them to work out their meaning without translating them to a programming language.	• Numerical algorithms are often the simplest to use in this case as their meaning is relatively easy to work out (e.g. find the highest, get the average and so on).	• Students are required to interpret the meaning of simple algorithms and also to correct deliberate mistakes when given the intended meaning.
Language Libraries	3.1.4	1st	• Three hours to examine some features of the particular programming language that will benefit students. For instance this could be built in functions for handling user input or commonly used data structures such as dictionaries.	• This section is completely dependent on the language used. These three hours can be used to prepare students more fully in how to utilise features of their language when completing the	

				controlled assessments.
		2nd	• See above.	
		3rd	• See above.	
External Databases (Text)	3.1.7	1st	• Discuss why it may be desirable to have data for a program(s) stored externally to the code. • Provide students with the code necessary to read data in and write data to text files. • Students should write two procedures that store a string to a text file and also reads the string in and prints the result.	• In the first lesson ensure that students can read and write to an external file. This may be most easily done using a common format such as CSV or could just be reading in line-by-line.
		2nd	• Practise with examples such as reading and saving high scores in a game or creating text by using random words from the English language.	• This lesson should require students to extract relevant information from external files and so should include basic parsing (such as each line containing a name, a comma and then a high score and the program tokenising these results). • Comprehensive lists of English words can be found at http://wordlist.sourceforge.net/
External Databases (SQL)	3.1.15, 3.1.15.1	1st	• Discuss the limitations of a text based (flat-file) database for more complex data. • Show, by example, how forming relations between tables of data can reduce the	• This could be done graphically with Velcro records stripped away from tables and linked

			amount of data that needs to be stored and increase the ease of updating data. • Students should practice creating their own related tables.	using coloured string and finally showing how this is done with primary and foreign keys.	
		2nd	• Provide students with data and work through examples to build on intuitive notion of SELECT, FROM and WHERE.	• You may want to recap on previous work with Boolean operators at the start of this lesson.	
		3rd	• Frame the previous work in terms of record, field, table, primary key, query, relationship, index and search criteria. • Interface with an SQL database using the students' programming language and practise forming queries, interrogating the database and iterating over the results.		• SQL queries can be run on MS Access but you may want to host an SQL database elsewhere; at the time of writing PythonAnywhere provides a free MySQL database that students could access over a network.
		4th	• Extend the SQL statements to include ADD and UPDATE and continue to use a programming language to interrogate and update a database.		
Networks	3.1.13	1st	• Students should identify how many computational devices they own that are not part of a network (this will probably be mainly microprocessor driven devices). Discuss the advantages of networking devices. • Use long pieces of string with messages 'hooked' on them to indicate the difference between ring, bus and star topologies. • Analyse the topologies for fault resilience, cost	• Students can create physical representations of the different topologies. The analogy may well break down when a 'switch' has to control all messages in a star network, this will require further	• It isn't necessary for students to have an in depth knowledge of how large networks, such as those used in schools, are comprised but it may help them to realise the composite nature of the Internet as being a

			and 'speed'.	explanation that the 'switch' is highly efficient at routing messages.	network of networks.
Client-Server Model	3.1.13.1	1st	• Use route tracing software to analyse the route an HTTP request takes to a web server; all students should repeat the request to see if the routes are all identical. • Illustrate HTTP in a simplified way to show how the handshaking process works. • Ask a school technician to show the servers running at a school and the jobs that the different servers perform. • Before the lesson set up a web server in the class room using an old computer and show students how access its web files over the local network. Illustrate the handshaking protocol with basic reference to HTTP.	• Most operating systems (including Microsoft Windows) have the ability to trace packages using route tracing command line tools. Check beforehand that students have the necessary permissions to access these tools. • The web server may be useful later if the students are going to complete the web based project for their controlled assessment.	
Client-Side Programming	3.1.13.2, 3.1.13.1	1st	• Provide students with HTML files including forms that require user input. • Students explore the use of HTML forms and customise the templates they have been given.	• This assumes some familiarity with HTML/CSS – if students are new to HTML/CSS then use an online tutorial (such as W3Schools) to cover the basics.	
		2nd	• Give students examples of JavaScript and get students to validate input and alter the HTML/CSS if the input is invalid.	• JavaScript is so universal as the language of client-side scripting that it makes sense to	

				introduce it in this context even if students haven't had any previous experience of it. Codecademy.com provides an excellent interactive tutorial approach to learning the basics of JavaScript.	
		3rd	• Further practice on user validation. • Explain the concept of robustness and how validating user input is a basic but essential form of helping to ensure programs maintain correct performance.	• Extend the examples from the previous two lessons and encourage students to test their input fields with out of range data, missing data or incorrectly typed data to ensure they are as robust as possible.	• You could build on the concept of robustness during the next extended homework which will involve user input through to database interrogation.
Server-Side Programming	3.1.13.2, 3.1.15.2	1st	• Allow students shell access to a web server and show examples of how server scripts can respond dynamically to requests, possibly requiring database interrogation. • A basic start is to use a server-script to embed the current time in milliseconds in a webpage. • Extend these tasks over the next four lessons to include scripts that interrogate and update a database and manipulate the returned HTML pages accordingly. • If your students intend to take the controlled assessment in web development they will probably also want to cover the use of cookies	• Many languages can be used for server-side processing and students shouldn't necessarily have to change to a language such as PHP to complete these tasks. • In order to complete these tasks students will need privileged access to a web server running the necessary language modules.	• Consult with the network technicians to decide the best way to create and manage a web server. This doesn't need to be a high-end machine and for the needs of this course an old computer sitting in the corner of the classroom should suffice.

			or sessions to enable restricted access and access to previous states.	• The support materials for the specimen controlled assessment could be used with both this and the previous lessons in client-side programming.
		2nd	• See above	
		3rd	• See above	
		4th	• See above	
Prototyping & Testing 3	*3.1.11.1, 3.1.12*	*hw*	• *Provide students with the HTML/CSS framework for them to work in and an empty external SQL database and give them a brief.* • *Encourage an iterative approach to developing a dynamic website that includes unit testing throughout.*	• *The final suggested extended homework task builds on the previous work on SQL databases and client and server side programming. Students could be given a brief and an HTML/CSS framework (the specimen controlled assessment materials are ideal).*
External Code Sources	3.1.14	1st	• Motivate the use of external code sources through web based libraries such as external fonts, scripting libraries and similar. • Discuss and analyse when students have used external code sources throughout this course and the advantages it provides along with possible problems.	• Fonts can be loaded into a web page using the `src` attribute in CSS. • JQuery could be downloaded and used in a web page to enhance the work already done on client side programming.

		2nd	• Students to return to their dynamic websites and integrate external tools such as Google Maps, reference to external fonts and so on.	• Students will have to be introduced to the Google APIs, this is a good starting point: http://designshack.net/articles/html/embedding-google-maps-into-a-web-page-a-beginners-guide/
Computer Technology in Society	3.1.8.1, 3.1.8.2 3.1.16	1st	• Introduce the concepts of safety critical and mission critical software and ask students to compile of list of such software. • Use a current issue in computing that requires a significant amount of hardware and processing, an example might be the plan to hold a national database on every Internet users' emails and web browsing history or the use of biometric checking at the UK border. • Discuss the hardware and processing necessary to support such as system. • Further discuss the need for reliability and robustness by discussing the implications for users if errors appeared in this system.	• Scanning journalistic articles prior to this lesson will help inform the topic.
		2nd	• Give a brief introduction to the history of GPS and its modern civilian use. • Students to critically discuss the reliance on GPS on some areas civilian life and the potential consequences if this signal was to be turned off.	• Encourage students to form evaluative judgements on the impact of GPS and any related issues.