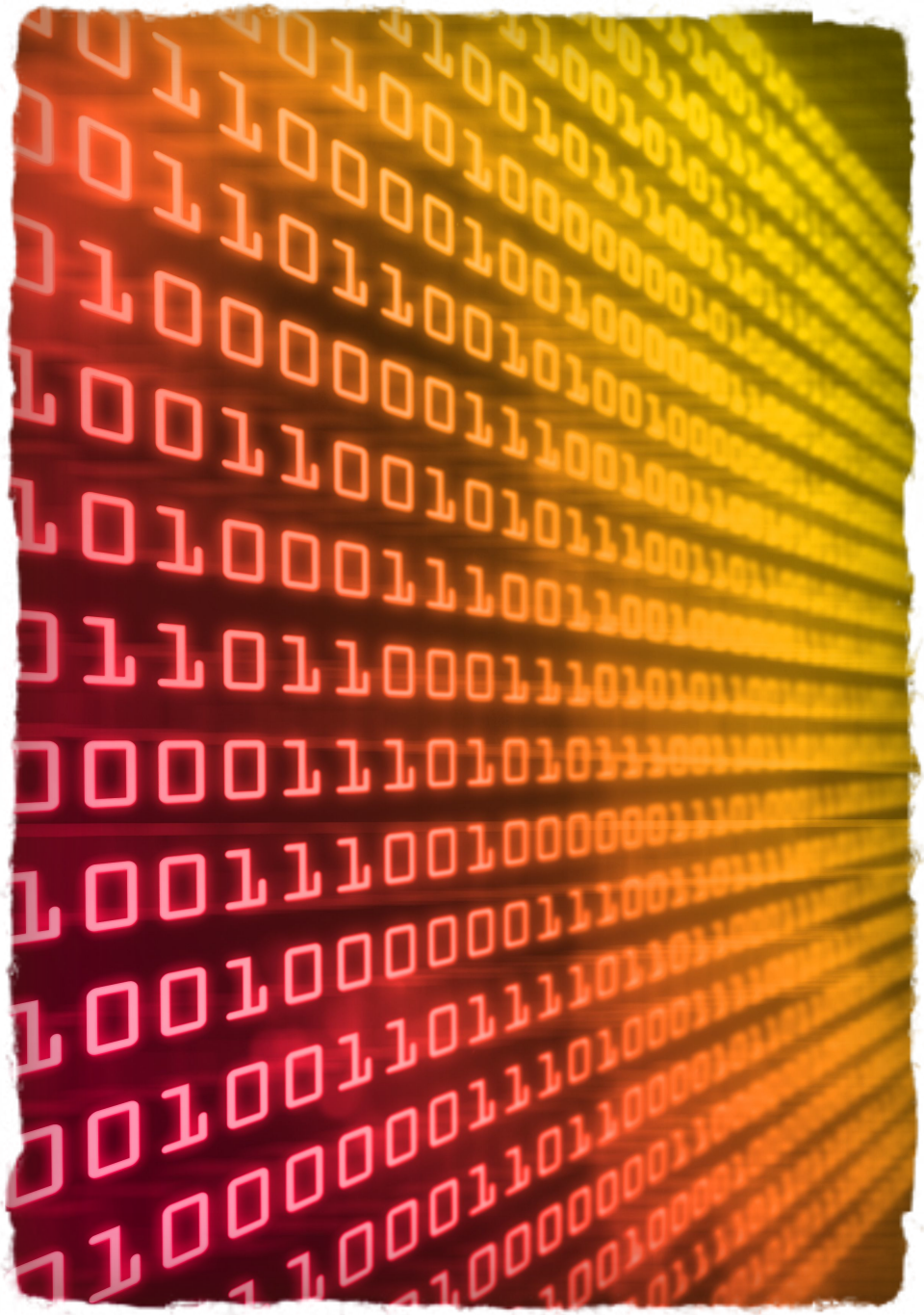
OCR GCSE Computing
An Unofficial Teacher's Guide



Mark Clarkson • Computing At School • December 2012

Preface

STUDENTS/PUPILS = This guide is about how to deliver the course. It does not have the answer to your homework / controlled assessment / test / exam in it. If you really want to read it, that's fine, but it'll be quite boring and won't help you cheat. Sorry.

This guide is written by a practising teacher who happens to be a member of Computing At School (CAS). The author has NO official capacity to speak on behalf of OCR, JCQ, DfE or any other official bodies and any advice should be treated as such. No liability will be accepted by CAS or the author if it all goes horribly wrong because anyone relied on this document.

This guide was written in December 2012, and refers to the OCR GCSE Computing specification and JCQ rules on Controlled Assessment at that time. Please check the current specifications, assessment documents and statutory guidelines.

The guide is split up into 4 sections:

- A451 - Written Examination
- A452 - Practical Investigation
- A453 - Programming Project
- Planning Delivery

A451 - Written Examination

The written exam is designed to test students' understanding of various topics related to computer science.

The specification splits the topics into the following sections:

- 2.1.1 - Fundamentals of Computer Systems
- 2.1.2 - Computing Hardware
- 2.1.3 - Software
- 2.1.4 - Representation of data in computer systems
- 2.1.5 - Databases
- 2.1.6 - Computer communications and networking
- 2.1.7 - Programming

There are a variety of tools, resources and bits of advice available at the CAS Online website (http://community.computingatschool.org.uk/) and at Teach ICT (http://www.teach-ict.com/). Below are a few ideas that I have used in the past and general thoughts about the content.

2.1.1 - FUNDAMENTALS OF COMPUTER SCIENCE

The topics in this section are quite difficult to pin down exactly, and the majority of the content I tend to cover at the end of the course. Issues over professional standards (e.g. policies) and ethical/environmental/legal considerations are easier to discuss once the students have some context to wrap these issues around.

The one exception would be the definition of a computer system. Starting by discussing whether a laptop, mobile phone, DVD player and washing machine each count as a 'computer' gives the students a familiar starting point.

2.1.2 - COMPUTING HARDWARE

This section is a good one to start with as the students will already be quite familiar with many of the concepts and terminology involved and so will find the learning curve a bit less steep early on.

A fun practical activity is to get some old, unused but preferably working computers and putting students in small groups with the task of disassembling and photographing the various components. If the computers go back together and boot up then the students will feel quite proud of themselves and less afraid / ignorant of the hardware. You could also use the same computers later on to explore different operating systems.

I usually start with input/output devices which are reasonably straightforward and familiar to both existing ICT teachers and students alike. Simple matching exercises or imagined scenarios are useful tools to draw out knowledge and understanding. It is worth spending a little time investigating the differences between impact (e.g. dot matrix), inkjet, laser and plotter type printers. The students aren't expected to have intricate knowledge of the workings, but should be aware of the pros and cons.

Secondary storage is also a fairly familiar topic, although students sometimes struggle a little to understand the difference between the device and the medium. Again, match exercises and imagined scenarios are very useful. Students need to be familiar with the 3 basic technologies and there are plenty of websites out there that will explain how CD drives work. If you have a spare hard drive or 5 then allowing the students to disassemble them is quite good fun - but watch out that they don't walk off with the magnets as they are extremely powerful.

The CPU is the topic I cover next, and looking at the ALU (arithmetic and logic unit), IAS (immediate access store - or 'registers'), Control Unit and Clock is generally fairly straightforward. There are several animations to demonstrate the inner workings of the processor and I sometimes (depending on the students) spend a little time using a Little Man Computer emulator to demonstrate the fetch-execute cycle (although only a basic awareness of this is required). Students should be aware of the consequences of increasing the number of cores, the clock speed and the cache size (see the next paragraph).

Next comes memory, and examining the cache (key phrase: "acts as a buffer between the CPU and main memory"), RAM and virtual memory is kept distinct from the comparison of RAM and ROM. Flash memory links back to the secondary storage discussion and students should finish this unit being aware of SSDs (Solid State Drives) and hybrid drives (part HDD, part SSD).

Binary Logic I take out of the hardware section. There's already a lot of content in this unit and so I give them an end of unit test and then look at logic gates separately.

The level of understanding required here is fairly low. Only 3 types of gate (AND, OR and NOT) need to be learned, the idea of a truth table and how to produce a truth table from a simple diagram combining two gates (e.g. NAND, NOR, an AND and an OR) as well as vice versa. Boolean algebra is not necessary at this level. The Teach-ICT website has a very good logic gates workbook that covers the topic in slightly more detail than is necessary, and once students have completed this they will be very well prepared.

2.1.3 - SOFTWARE

This section is quite short, but you could easily expand it as it is useful, interesting and relevant to the students.

The short version simply splits into operating systems (what are the main functions), utility programs (what are the main examples) and applications (compare bespoke vs off the shelf and open source vs proprietary).

If you have some spare PCs or an installation of VirtualBox (free), the students can install their own Linux operating system (Ubuntu is user friendly and works very well, Fedora is more challenging and, frankly, more nerdy).

2.1.4 - REPRESENTATION OF DATA IN COMPUTER SYSTEMS

This is a key section and one that the students might find challenging in places - but they will generally be looking forward to finding out what this 'binary' stuff is all about.

Bits, nibbles, bytes, etc... is fairly straightforward, but doesn't mean a lot to start with. I usually start with binary - denary conversion. There isn't room here to go into the strategies, but there are plenty of websites and videos out there to demonstrate techniques here.

Hexadecimal conversion typically causes more of a challenge as the need for hexadecimal is a little less obvious than the need for binary. Examining colour codes and MAC addresses demonstrates the practical use of hexadecimal and students can also examine and even edit text files using a hex editor to try and understand why it all works (rather than simply how).

Character sets are pretty straightforward to teach. Many students will be familiar with pressing ALT-0246 for an umlaut (ü), or similar for French/Spanish accents. Looking at an ASCII table, students are quickly able to look up the correct conversion and the only tricky bit is getting students to accept that adding 1 bit to the character set doubles the number of available characters (i.e. 7 bits = 128 characters, 8 bits - 256 characters).

Exploring images can be quite fun. Starting with black and white images, the students can convert a bit pattern to a monochrome image and vice versa. The idea of using 2 bits to represent 4 colours, 3 bits for 8 colours, etc. can then be brought in. There are some very good spreadsheet and web-based tools that will allow students to convert images to binary and vice versa, experimenting with colour depth and different colour sets. Most image editing applications will also allow students to export GIF images using a variety of colour depths.

Image dimensions are straightforward, but resolution can be more difficult if you're not careful as the meaning of resolution can be different depending on the context (e.g. monitor resolution typically refers to the number of pixels rather than the true resolution). A key phrase is "the density of pixels" and comparing a 72dpi image (web-standard) with 300dpi (print-standard) and other resolutions can help.

Students should be able to understand what metadata is stored in image files, and looking at the properties of the file can be helpful here.

To investigate how sound is stored, students can open an MP3 file in Audacity and zoom right in to see the value of each sample. Changing the sample rate of an existing song also changes the playback speed, which is not ideal, but by exporting songs from iTunes with different settings you can give students access to the same music file at 44, 16 and 8kHz so that the link between sample rate and sound quality can be investigated.

The issue of colour depth in images is analogous to the issue of bit depth in audio files. Where a 16 bit-per-pixel image will have less accuracy than a 24 bit-per-pixel image, the same is true of a 16 bit-per-sample vs 32 bit-per-sample audio file. The existing software programs tend to only offer option from 16 bits upwards, making it hard to show what a low bit-depth audio file will sound like, but the concept is not too difficult for students to grasp.

For both audio and images, the issue of compression is mentioned in the specification. While it is worth briefly looking at the difference between a raw bitmap or sound file and a compressed jpg or mp3 file I tend to leave the detailed work on compression until the section on websites.

The final part of this unit is looking at how instructions are stored. This is another case where the Little Man Computer can be a useful illustration. Here we can see how instructions are converted to binary code and stored for later retrieval, decoding and execution (fetch-decode-execute cycle).

2.1.5 - DATABASES

Unlike traditional GCSE ICT courses, there is not much emphasis on databases in this specification. I find that practical work helps the students to grasp the content and context of the topic and so, while there is no practical assessment, I do generally resort to using Access (amongst other tools) to present the learning.

While students are expected to learn about records, tables, fields, validation and simple queries, there is a greater emphasis on how databases actually work. The explicit reference to a DBMS (Database Management System) as separate from the data (in this case an MDB file) is different to an ICT specification and students are explicitly expected to understand how and why related tables are used and how to use primary and foreign keys to accomplish this.

A simple 2 or 3 table structure with 1-to-many relationships is quite sufficient and I get students to enter various redundant data before discussing how we could improve efficiency by having all of the person details in one table and reference to it in the second table (be it subject grades, purchases, records of criminal convictions, etc).

It may well be worth exploring simple PHP with a MySQL database if you feel up to it, but the SQLZoo website offers an excellent set of online tasks examining simple and complex queries using SQL to pull out the required data. I tend to skip the countries database and move on to the Nobel Prizes tasks.

2.1.6 - COMPUTER COMMUNICATIONS AND NETWORKING

This section is largely split into 2 parts. The first deals with networks (LANS and WANS), the second with Internet specific issues. Much of the content is similar to previous ICT specifications - LAN vs standalone, LAN vs WAN, toplogies, etc. Here, though, IP and MAC addressing are specifically included, and it is interesting for students to explore the IPCONFIG, NSLOOKUP, WHOIS and TRACERT tools at the command line. If students do not have access to the command line then there are various websites which will allow students to run WHOIS and Traceroute type activities.

The topic of Client-Server vs Peer-Peer arises and this is a more technical topic than some teachers may be used to, but between Google, Wikipedia and Youtube there are plenty of resources out there to explain it in simple-to-understand terms.

The section on the Internet suggests that students should be aware of, but not necessarily able to code in, HTML "and its derivatives" (by which I take it to mean CSS). As with databases, the students can contextualise this best through practical activities, and the use of Mozilla Thimble can be very helpful as a text-based coding environment for web page creation.

IP makes a return, along with DNS this time, and it is useful for students to understand how a packet gets from source to destination. Again, there are plenty of YouTube videos to support this.

Compression also returns, having appeared previously in the binary representation section of the specification, and students often struggle to comprehend lossy vs lossless compression. A good tip is to get students to take an image file and try zipping it up, changing the resolution, changing the colour depth and so on = making a new copy from the original each time. By tracking the file size savings and comparing the quality of the image once reconstituted (e.g. unzipped, with the resolution enlarged once more, with the greater bit depth) the students can easily see how damaging some compression techniques can be - but also how big the file size savings can be.

2.1.7 - PROGRAMMING

While the majority of this topic will be covered through practical activities, both before and during the controlled assessment, there are some issues that students will need to understand on a specifically theoretical level.

The issue of translators (assemblers, compilers and interpreters) is not one that is obvious unless you have programmed in a number of languages. Given that students are unlikely to have had the time to experience this for themselves it must be learned in a more abstract setting. Again, though, this is where the Little Man Computer might come in handy.

Students should be able to write about the different data types as well as use them, and similarly for the basic constructs of sequences, selection (IF) and iteration (FOR and WHILE).

Finally, students should know the difference between syntax and logic errors as well as understanding why and how to test properly.

On a personal note, I've really enjoyed having the opportunity to teach like a proper teacher - with theory lessons, exercise books, notes and all the things that our colleagues in History, Science, English, et al probably find frustrating.

A452 - Practical Investigation

The Practical Investigation controlled assessment is intended to take 20 hours and should be carried out under controlled conditions.

OCR have a range of possible tasks, available via Interchange. Without going into details here, there are some options which tie in nicely with aspects of the theoretical understanding. I like these options as they add to the narrative for the students.

There are some options that are wholly separate from the specification, and I like these because they allow the students to learn about something totally different and new.

Either way, you must use one of the set tasks.

OCR recommends that students spend :

- Around 2 hours introducing the topic and the way in which students will tackle the tasks (how to document their work, where they can go for research, what kinds of evidence to produce, etc. This is classed as "informal supervision" and while you have to keep an eye out for plagiarism, there are no special requirements and group work is allowed.
- Around 6 hours researching (this is classed as "limited supervision"). This work can be carried out away from lessons, and so it is not necessary to prevent access to the Internet, nor to stop students from talking to each other - in fact group work may well be encouraged. The only things you can't do are to give direct feedback on work, practise the task or provide templates/model answers.
- Around 12 hours completing the task. This is classed as "formal supervision". Work should be collected in between sessions (we use dedicated accounts that are only active during lessons) and candidates should be monitored at all times to make sure that work is their own.

In practise, the guidelines above are only that. The work can be done in any order and you don't need to document the timings or sessions involved. It is useful to keep a separate register so that you can make sure students have had sufficient time to get the work done, but allowing a student 21 hours is not a problem.

One of the things that surprised me most when I started the course was just how practical the practical investigation really is. Many of the tasks available are essentially programming tasks, with some research required to explain the context.

Lets take as an example, a task that involves a simple SQL database (largely because there is currently no such task!).

Lets pretend the tasks are to

1. Run a pre-written SQL query based on 1 table and describe the output
2. Explain the logic behind the SQL query and how/why it produces the output it does.
3. Add to the query to make it display data from 1 extra field (plan, write and test the code)
4. Write a new query on the same table (plan, write and test the code)
5. Write a new query that reads data from 2 related tables (plan, write and test the code)
6. Write an evaluation of your solutions
7. Write a conclusion about the effectiveness of SQL as a method of interrogating databases

The work does not need to be completed all in one go - you don't have to simply say "start", and then 20 lessons later say "stop". You can't give the students the same task to complete in lessons, but you can provide guidance to the whole class that will point them in the right direction. So I might hand out the controlled assessment materials, spend a little time (1 lesson) going over the rules and regulations, the kinds of research and evidence that are required, etc.

For lessons 2 to... 4? I might send students off to SQLZoo and similar sites to work together and try to understand SQL. From the specification, students can work together, work at home and more. I wouldn't want the students to have the task in front of them at this point as I want them to focus on more generic learning, but I might provide them with a list of key words (SELECT, FROM, WHERE). I would encourage the students to stick to simple, 1-table queries and would suggest that they make notes. I would not be allowed to respond directly to those notes but I would feel comfortable looking over students' shoulders while they worked and, where there were common misconceptions or omissions I would use that to inform the way I delivered the next lesson.

The next few lessons would focus on questions 1 and 2. I would give students suggested deadlines (e.g. "Make sure you're finished both questions by the end of the second lesson"), but I wouldn't be able to give specific feedback to individuals. I couldn't show them model answers, but I might wave an exemplar from my last cohort briefly in front of them (for a few seconds) **provided it was for a different task**. The idea here would be to show them the style of response that is expected but **without showing them model answers to that question**.

We would then pause and I would give the students a totally different SQL task - perhaps one of the tasks from SQLZoo. I would get them to plan, code and test their solution. I wouldn't give direct feedback to students, but I would give generic feedback or advice to the whole class.

I would then return to tasks 3 and 4, safe in the knowledge that the students have researched the skills and understanding required to solve the problem and that I have given them enough support to allow them to document it properly.

After that, another break to research how to query on related tables before returning to task 5.

If I had weaker students who were struggling, I would prefer that they stuck to tasks 3 and 4 for now.

With a few lessons left I would encourage students to switch to the evaluation and summary tasks. I would rather students completed tasks 3 and 4 followed by the evaluation and skipped task 5 rather than trying 3, 4 and 5 but missing the evaluation.

When it comes to marking - the OCR focus is very much a case of "give the student the grade they deserve - then justify it". The A452 mark scheme is deliberately vague because it applies to any and all of the available tasks and also allows you to reward good work without having to make the students jump through hoops.

For marking I found it easiest to print out all of the work (collated into one Word document per student), attack it with a highlighter and a pen and then scan in the finished version. This doesn't take as long as it sounds and because of the nature of the controlled assessment it only has to be done once.

Mark Scheme Guidance:

Caveat - this is my understanding only

Practical Investigation

Does the document make sense, is it clear, has the student clearly attempted to do what was asked? Is there good evidence to demonstrate what the student has understood? Annotated screenshots (a few - not death by screenshot)? Clear sentences? Technical language? Planning documents? An informed summary?

Efficient and Effective Use of Techniques

Has the student used SELECT, FROM and WHERE statements, used a logical test and interrogated a relational database correctly? Some credit for trying, some more for succeeding in a reasonably efficient manner (remember, this is still only GCSE level).

Technical Understanding

Has the student used, and explained, technical terms? Does the summary demonstrate a sound technical understanding? Has the evidence of the effectiveness of SQL gathered during the research stage been collected, understood, used?

Conclusions and Evaluation

Is the conclusion appropriate and accurate? Has each task been fully tested and evaluated, with pros and cons? Spelling, punctuation and grammar.

It's open to interpretation, and there is some overlap, but good students will come out well and weaker students will do less well, while still being able to access appropriate grades.

For grade boundaries I estimated students using the following, which seemed to bear out:

90% - 41/45 - A*
80% - 36/45 - A
70% - 32/45 - B
60% - 27/45 - C
50% - 23/45 - D
40% - 18/45 - E
30% - 14/45 - F
20% - 9/45 - G

A453 - Programming Project

The programming project runs in very much the same way as the practical investigation.

The advice from OCR is exactly the same, and there are multiple tasks to choose from - but students must complete all 3 questions from the same overall task.

Students can use any language they like, as long as they can complete the task. They can also swap and change environments so they **could** complete question 1 with Scratch, question 2 with Python and question 3 with Visual Basic if they wanted. That's not necessarily the most straightforward route, but it is acceptable.

Making up another hypothetical task:

- Create a program that will generate two random numbers - one that maps to a suit (hearts, diamonds, clubs or spades) and one that maps to a card (Ace, 2, 3, ... Jack, Queen, King)
- Create a program that will generate a random card and then ask the user to go Higher, Lower or Quit. If the player chooses Higher the next card must be of a higher value or the player is out. Likewise for Lower.
- Extending the previous program, the user will select a trump suit at the start of the game. If the card is a trump suit card then the game continues regardless of the card's value. The program will keep score and will save the score to a highscore file. The user will also be able to display the current highscore file.

As you can see, the tasks feed into each other and get increasingly complex. The final program is actually quite complex, but all students should be able to complete the first task.

As with the A452 project I would get students to investigate the generation of random numbers and mapping them to a character set (which is all this is, in effect) before setting them off on the actual task.

Similarly, I would stop and focus on loops before sending students back to tackle part 2.

Finally, I would expect to spend some time looking at file handling before students tackled part 3.

OCR have stated quite clearly that there is no reason to prevent students having access to the Internet during controlled assessments and although it is important to watch for plagiarism, in practise most programmers will occasionally need to refer to online APIs or even searching through existing forum posts for help with specific problems or syntax (e.g. to map a random number to a card value might be better with a CASE rather than an IF statement). Students should obviously not be searching for direct, complete solutions.

I get students to tackle each task in the following order:

Planning

- What is required (summarise the problem in your won words)
- Approach (e.g. "I will use a random number generator and IF/ELSE IF/ELSE statements to turn the random numbers into a suit and value)
- Design (a flowchart, pseudocode or other evidence of algorithm planning)
- Testplan & Success Criteria (test data should always be prepared before the program is written as this is separate to the code and it is too easy to test for what you have written rather than against the original task)
- Variables (list the variables you plan to use, including data types)

Development

- Development diary (either a lesson-by-lesson summary of where the program is at OR a short diary entry whenever you hit a problem, error or milestone - it is important to provide evidence of testing during the development phase, ideally showing systematic and planned testing of individual parts)
- Completed, commented & annotated code

Testing

- Evidence that the test plan has been completed

Evaluation

- The student's own opinion of the solution - both in terms of completeness and efficiency (how many variables, reuse of code, clarity)

In terms of marking the work, the mark scheme is, again, open to some interpretation.

Use of Programming Techniques

Has the student used variables, assignment, selection (IF), iteration (LOOPS), arrays, file handling, etc. AS APPROPRIATE? There's no need to invent more complexity to justify the use of an array, for example, but if the problem leads to a natural solution that uses an array then the student with full marks will hopefully have done so.

Most of my students last time through did not use methods/functions at all, but many achieved full marks in this band. I do believe it is better to encourage students to work with methods and functions, however.

Efficient Use of Programming Techniques

Do the programs work? You can award marks for a program that generates a random number and compares the values but doesn't examine the suit as the program has been partially solved - even if the program doesn't run.

Is the program efficient? A good program will be fairly short and get things done in a simple way. A weaker student might struggle and bumble along a bit - leaving some disorganised code behind them.

Design

Has the student analysed the task and justified an approach? Have they designed the algorithm(s)? Have they decided on testing and success criteria before coding? Have they planned suitable variables with suitable data types?

Development

Has the student demonstrated how they arrived at their solution? Have they shown that they tested their program **during** development? Is the code well organised, with sensible variable names and comments?

Testing

Is there a thorough test plan, with evidence of whether each test is met? Is there an evaluation against the success criteria? Are technical terms and good SPaG used throughout?

Planning Delivery

In many ways, this part should be the first, and most important part of this document - but without the context of how the work will be assessed it is difficult to formulate a clear plan.

PROGRAMMING

The first thing to realise is that programming is a difficult **and different** skill. Students are not used to struggling, solving their own problems or repeatedly failing. They have spent 10 years or more learning to give the *correct* answer, and if they didn't know it, then to learn it.

In programming, students need to try, to fail, to realise that the sky has not fallen on their heads and to try something different. It is incredibly difficult to watch a student struggle and not dive in with the answer, but the process, the techniques and the strategies are far, far more important than simply getting a program that does what you want.

To that end, I make sure my students do some programming every week, and do some theory every week (except during controlled assessment).

For the first half-term we do nothing but logic problems without going near a computer (search CAS Online for good examples) and in the second half-term we look at basic Python programming, covering input, output, variables, assignment, if statements and loops (WHILE and then FOR). Regular programming homework tasks and paired programming challenges during lessons ensure that students keep practising, and in a safe and secure environment where it is OK to fail. In fact failure must be celebrated as it means the student in question has moved one step closer to a solution.

It is vital that students also practise planning their programs. Most professional programmers will avoid flowcharts like the plague, but for a beginner (and especially one who is used to visual environments such as Scratch and Alice), flowcharts can be a very good way of planning an algorithm.

Students should also evaluate their programs, as this is a very useful skill that is drawn upon in the controlled assessment.

When setting a programming task, never give students a blank canvas.

First, solve the problem yourself - write the code. Then remove bits or break bits so that the students have a skeleton to build on. Maybe print each line out, one by one, cut them up and get the students to arrange them into a working program. Maybe replace key lines with dummy text and have students finish them off. In the say way that English and History teachers will give students a writing frame, give them a coding frame.

Provide them with a reference guide - how to write an IF statement, the syntax for a FOR loop, etc. Students are not banned from using these in controlled assessments as long as the example code is not directly as it should be in the program.

THEORY

With two lessons a week, if I am spending one lesson on programming, the other is on theory. I cover hardware in the first half term, and then binary/boolean logic in the second. After Christmas we look at software and then a topic relating to the chosen A452 assessment.

I suspect that I am not alone in finding the theory topics quite challenging in the sense that I am used to teaching students practical skills. I deliberately plan time away from the computers for theory lessons and have students working in exercise books or lined paper (kept in a ring binder). I've had to talk to colleagues from other departments to find good strategies and techniques for delivering a 'traditional' lesson - but I've enjoyed it and the students appreciate the need to learn things in the same way they do in Science, Geography, etc. For them it is no different, while for me it is very much so.

CONTROLLED ASSESSMENT

I schedule the A452 assessment in Year 10, straight after February half-term. Realistically I need at least 10 weeks, probably more, which should take me up until May half-term. This gives me the last half term to get back to some programming skills in Python (my language of choice) and also allows a little wiggle room should there be a prolonged absence in there somewhere.

The A453 assessment I start early in September of Y11, taking it most of the way to Christmas, at which point we can return to theory.

OTHER OPTIONS

One of the most commonly asked questions is that of which language to teach to the students.

The slightly pious answer is that you should be teaching principles, not a language.

A more practical answer is to find a language that you feel comfortable with and have some (preferably local) support for and use that. Almost any modern programming language will allow you and allow your students to complete the tasks required - VB, C++, Python, Java, BASIC, Delphi, JavaScript...

All have pros and cons, and Python & VB appear to be the most commonly chosen languages at the moment, meaning that there are lots of resources out there to draw upon.

Anything else I've missed? I'm sure there is plenty.

If you want to find some help or advice, you could try any or all of the following.

If you see a mistake, omission or other problem with this guide then please do get in touch with me (mwclarkson@gmail.com / @mwclarkson).

Other sources of support:

- CAS Online
- Your nearest CAS Hub or Master Teacher
- ComputingPlusPlus

Good luck, and most of all - have fun!

Mark Clarkson

December, 2012