Scheme of Work

Cambridge International AS & A Level Computer Science

9608

For examination from 2017

Cambridge Advanced

Contents

Introduction

This scheme of work provides ideas about how to construct and deliver a course. The syllabus has been broken down into teaching units with suggested teaching activities and learning resources to use in the classroom. This scheme of work, like any other, is meant to be a guideline, offering advice, tips and ideas. It can never be complete but hopefully provides teachers with a basis to plan their lessons. It covers the minimum required for the Cambridge AS and A Level course but also adds enhancement and development ideas on topics. It does not take into account that different schools take different amounts of time to cover the Cambridge AS and A Level course.

Recommended prior knowledge

Candidates beginning this course are not expected to have studied computer science or ICT previously.

Outline

Whole class **(W)**, group work **(G)**, pair **(P)** and individual activities **(I)** are indicated, where appropriate, within this scheme of work. Suggestions for homework **(H)** and formative assessment **(F)** are also included. The activities in the scheme of work are only suggestions and there are many other useful activities to be found in the materials referred to in the learning resource list.

Opportunities for differentiation are indicated as **basic** and **challenging**; there is the potential for differentiation by resource, length, grouping, expected level of outcome, and degree of support by teacher, throughout the scheme of work. The length of time allocated to a task is another possible area for differentiation.

Guided learning hours

Guided learning hours give an indication of the amount of contact time teachers need to have with learners to deliver a particular course. Our syllabuses are designed around 180 h for Cambridge International AS Level and 360 h for Cambridge International A Level. The number of hours may vary depending on local practice and the learners' previous experience of the subject. The table below gives some guidance about how many hours are recommended for each topic.

Topic	Suggested teaching time (recommended)
1.1 Information representation	10 hours
1.2 Communication and Internet technologies	15 hours
1.3 Hardware	15 hours
1.4 Processor fundamentals	15 hours
1.5 System software	15 hours
1.6 Security, privacy and data integrity	10 hours
1.7 Ethics and ownership	10 hours
1.8 Database and data modelling	20 hours
2.1 Algorithm design and problem-solving	20 hours
2.2 Data representation	15 hours
2.3 Programming	25 hours

2.4 Software development	10 hours
3.1 Data representation	15 hours
3.2 Communication and Internet technologies	15 hours
3.3 Hardware	20 hours
3.4 System software	20 hours
3.5 Security	15 hours
3.6 Monitoring and control systems	20 hours
4.1 Computational thinking and problem-solving	20 hours
4.2 Algorithm design methods	15 hours
4.3 Further programming	25 hours
4.4 Software development	15 hours

Teacher support

Teacher Support (https://teachers.cie.org.uk) is a secure online resource bank and community forum for Cambridge teachers, where you can download specimen and past question papers, mark schemes and other resources. We also offer online and face-to-face training; details of forthcoming training opportunities are posted online.

This scheme of work is available as a PDF and an editable version in Microsoft Word format; both are available on Teacher Support at https://teachers.cie.org.uk. If you are unable to use Microsoft Word you can download Open Office free of charge from www.openoffice.org.

Resources

The up-to-date resource list for this syllabus, including textbooks endorsed by Cambridge, is listed at www.cie.org.uk and Teacher Support https://teachers.cie.org.uk.

Endorsed textbooks have been written to be closely aligned to the syllabus they support and have been through a detailed quality assurance process. As such, all textbooks endorsed by Cambridge for this syllabus are an ideal resource to be used alongside this scheme of work as they cover each learning objective.

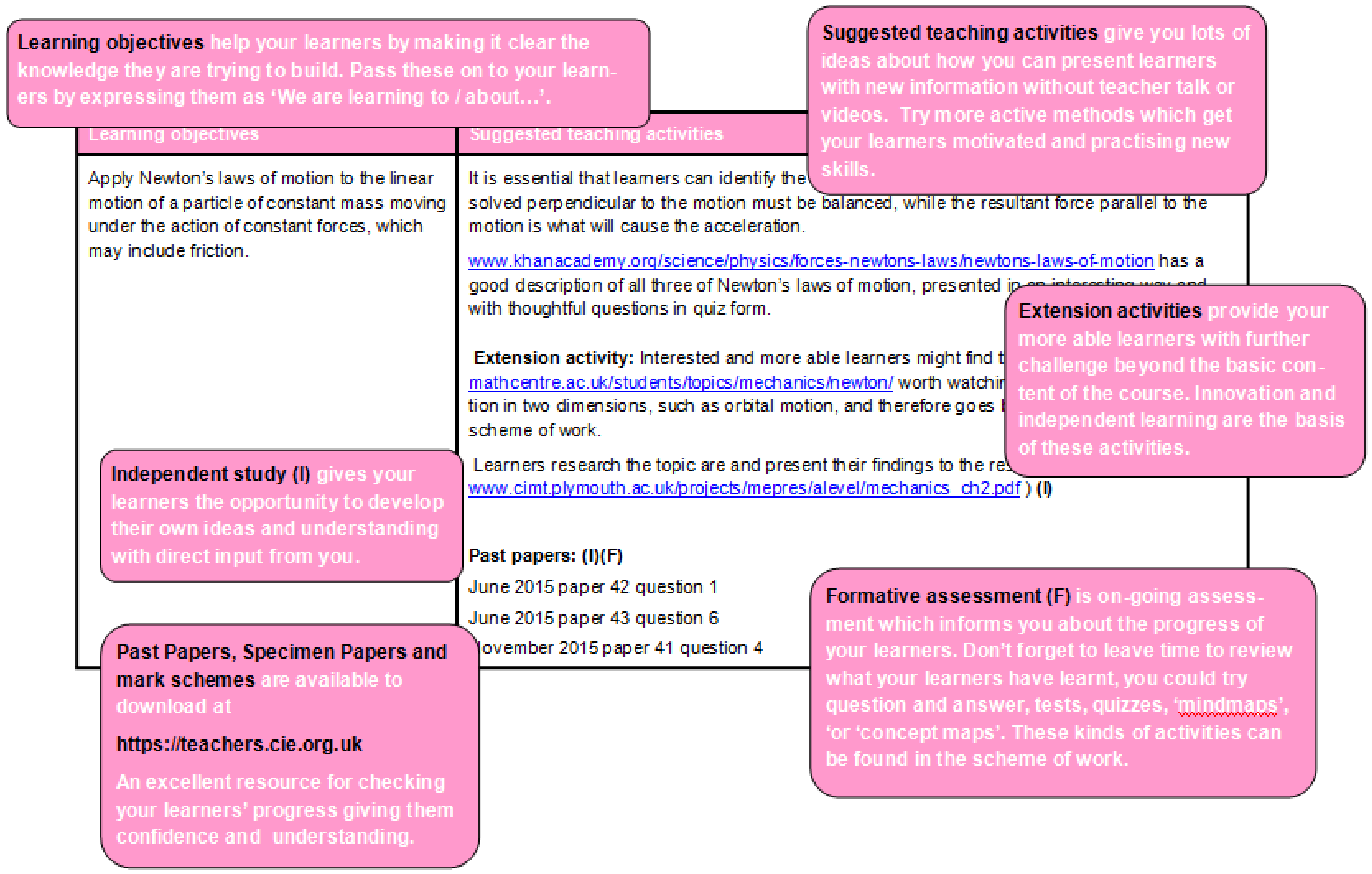
Websites and videos

This scheme of work includes website links providing direct access to internet resources. Cambridge International Examinations is not responsible for the accuracy or content of information contained in these sites. The inclusion of a link to an external website should not be understood to be an endorsement of that website or the site's owners (or their products/services).

The particular website pages in the learning resource column of this scheme of work were selected when the scheme of work was produced. Other aspects of the sites were not checked and only the particular resources are recommended.

How to get the most out of this scheme of work – integrating syllabus content, skills and teaching strategies

We have written this scheme of work for the Cambridge International AS & A Level Computer Science (9608) syllabus and it provides some ideas and suggestions of how to cover the content of the syllabus. We have designed the following features to help guide you through your course.



Learning objectives help your learners by making it clear the knowledge they are trying to build. Pass these on to your learners by expressing them as 'We are learning to / about…'.

Suggested teaching activities give you lots of ideas about how you can present learners with new information without teacher talk or videos. Try more active methods which get your learners motivated and practising new skills.

Learning objectives	Suggested teaching activities
Apply Newton's laws of motion to the linear motion of a particle of constant mass moving under the action of constant forces, which may include friction.	It is essential that learners can identify the solved perpendicular to the motion must be balanced, while the resultant force parallel to the motion is what will cause the acceleration. www.khanacademy.org/science/physics/forces-newtons-laws/newtons-laws-of-motion has a good description of all three of Newton's laws of motion, presented in an interesting way and with thoughtful questions in quiz form. **Extension activity:** Interested and more able learners might find t mathcentre.ac.uk/students/topics/mechanics/newton/ worth watchi tion in two dimensions, such as orbital motion, and therefore goes scheme of work. Learners research the topic are and present their findings to the re www.cimt.plymouth.ac.uk/projects/mepres/alevel/mechanics_ch2.pdf) (I) **Past papers: (I)(F)** June 2015 paper 42 question 1 June 2015 paper 43 question 6 ovember 2015 paper 41 question 4

Extension activities provide your more able learners with further challenge beyond the basic content of the course. Innovation and independent learning are the basis of these activities.

Independent study (I) gives your learners the opportunity to develop their own ideas and understanding with direct input from you.

Past Papers, Specimen Papers and mark schemes are available to download at

https://teachers.cie.org.uk

An excellent resource for checking your learners' progress giving them confidence and understanding.

Formative assessment (F) is on-going assessment which informs you about the progress of your learners. Don't forget to leave time to review what your learners have learnt, you could try question and answer, tests, quizzes, 'mindmaps', 'or 'concept maps'. These kinds of activities can be found in the scheme of work.

1.1 Information representation

Syllabus ref	Learning objectives	Suggested teaching activities
1.1.1 Number representation	• show understanding of the basis of different number systems and use the binary, denary and hexadecimal number system • convert a number from one number system to another	Demonstrate a step-by-step example showing the conversion from binary to denary and back from denary to binary. There are various methods that can be used to do this. **(W)** Demonstrate a step-by-step example showing the conversion from binary to hexadecimal and back from hexadecimal to binary. There are various methods that can be used to do this. **(W)** Provide learners with further questions for each and ask them to carry out the conversion. Another way to check learners understanding is to give them the answers to some conversions and ask them to check if they are correct. **(I)** **Resources:** A step-by-step explanation of how to convert from decimal to binary: http://courses.cs.vt.edu/~csonline/NumberSystems/Lessons/DecimalToBinaryConversion/index.html Notes on hexadecimal: http://courses.cs.vt.edu/~csonline/NumberSystems/Lessons/HexAndOctalNumbers/index.html An interactive binary number conversion test game: www.pwnict.co.uk/binaryGrid/index.html Comprehensive notes for binary and hexadecimal with exercises: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Binary_number_system

Syllabus ref	Learning objectives	Suggested teaching activities
		A video of a lecture on binary numbers (11:40 min): www.youtube.com/watch?v=biqp0HjJmfk Class activities to introduce binary numbers: http://csunplugged.org/binary-numbers A game to test learners' binary number conversion skills: http://forums.cisco.com/CertCom/game/binary_game_page.htm

Syllabus ref	Learning objectives	Suggested teaching activities
	• express a positive or negative integer in 2's complement form	Demonstrate, with board work, the use of 2's complement to represent positive and negative numbers. Stress how to represent both positive and negative numbers because many learners often only consider the use of negative numbers. This may be done via sign-and-magnitude and 1's-complement representations to show learners the reason for 2's complement (difficulty of arithmetic, two representations of zero), although questions will not be asked on these other representations. **(W)** Another way is to explain it in terms of a 'milometer' turned backwards past zero. In decimal, a milometer showing 0000 when turned back 1 mile would show 9999. A binary milometer would show 1111. This represents −1. Let learners work out what the milometer would show when turned back 2, 3 etc. **(I)** Check that learners can recognise whether a binary number is positive or negative. Check that learners can find a rule to recognise even numbers (positive and negative). Make sure that learners know they need a specified number of bits to represent signed integers. This means leading zeros for positive integers. **Resources:** Notes on 2's complement: http://courses.cs.vt.edu/~csonline/NumberSystems/Lessons/TwosComplement/index.html Notes and exercises for two's complement showing two different methods of conversion (binary subtraction not required): http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Two%27s_complement
		Sequence of two videos of very detailed explanation (with background) of how to store negative integers: www.youtube.com/watch?v=Ys_t6iSjboM (17:05 min) www.youtube.com/watch?v=hksGdVX5NBQ (12:38 min)

Syllabus ref	Learning objectives	Suggested teaching activities
	• show understanding of, and be able to represent, character data in its internal binary form depending on the character set used (Candidates will not be expected to memorise any particular character codes but must be familiar with ASCII and Unicode.)	Provide leaners with a table example of a character set, such as ASCII. There are many examples available on the internet. Explain the need for character sets and how each character is assigned a code. **(W).** Give learners a message in binary to decode using the character set. Then get learners to code a message for another class member using the character set. **(I) (P)** **Resources:** Comprehensive notes and exercises for ASCII: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/ASCII Comprehensive notes and exercises on Unicode: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Unicode
	• express a denary number in Binary Coded Decimal (BCD) and vice versa • describe practical applications where BCD is use	Provide learners with a worksheet containing codes in binary, hexadecimal and BCD to be converted into denary. Also provide conversions from denary values in both number bases and BCD (include how many bytes would be required). Wikipedia notes ('Basics' only required) gives a good explanation why this representation is significant in Computer Science. **(I)** **Resources:** Notes on BCD including practical application: http://en.wikipedia.org/wiki/Binary-coded_decimal

Syllabus ref	Learning objectives	Suggested teaching activities
1.1.2 Images	• show understanding of how data for a bitmapped image is encoded • use the terminology associated with bitmaps: pixel, file header, image resolution, screen resolution • perform calculations estimating the file size for bitmapped images of different resolutions • show understanding of how data for a vector graphic is represented and encoded • use the terminology associated with vector graphics: drawing object, property and drawing list • show understanding of how typical features found in bitmapped and vector graphics software are used in practice • justify where bitmapped graphics and/or vector graphics are appropriate for a given task	Get your learners to draw a vector graphic of simple shapes from mathematical formulae (e.g. a circle given the centre co-ordinates and the radius, the colour of the line etc.) and then draw a bitmap of a circle, colouring in 'pixels' on graph paper. **(I)** Discuss with learners what would need to be done to enlarge each image. **(W)** Ask learners to look at a piece of vector graphic software. They should identify the features available to create vector graphics. **(P)** Discuss with learners how these features might be used in practice. **(W)** **Resources:** Introduction: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Images Detailed notes and exercises on bitmaps: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Bitmaps Detailed notes and exercises on vector graphics: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Vectors Notes and exercises on differences between bitmaps and vector graphics: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Comparison_between_vector_and_bitmaps Classroom activity: http://csunplugged.org/image-representation

Syllabus ref	Learning objectives	Suggested teaching activities
1.1.3 Sound	• show understanding of how sound is represented and encoded • use the associated terminology: sampling, sampling rate, sampling resolution • show understanding of how file sizes depend on sampling rate and sampling resolution • show understanding of how typical features found in sound-editing software are used in practice	Explain the representation of sound to learners (see notes in wikibook). **(W)** Learners should then do the exercises in the last of the wikibook pages for this topic**. (I)** **Resources:** Comprehensive notes on sound: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Sounds Notes on analogue and digital sound: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Analogue_and_digital Notes and exercises on digital sound files: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Sampled_sound

Syllabus ref	Learning objectives	Suggested teaching activities
1.1.4 Video	• Show understanding of the characteristics of video streams: - the frame rate (frames/second) - interlaced and progressive encoding - video interframe compression algorithms and spatial and temporal redundancy - multimedia container formats	Get your learners to create a guide for their peers that informs them about the characteristic of video streams. **(P)** **Resources:** Notes on characteristics of video streams (section 2): http://en.wikipedia.org/wiki/Video Notes on multimedia container format: http://en.wikipedia.org/wiki/Multimedia_Container_Format Background information of streaming: http://en.wikipedia.org/wiki/Streaming_media

Syllabus ref	Learning objectives	Suggested teaching activities
1.1.5 Compression techniques	• show understanding of how digital data can be compressed, using either 'lossless' (including runtime encoding – RTE) or 'lossy' techniques	Give learners access to the compression techniques notes. Although not part of the syllabus, the Nyquist theorem is a useful starting point for discussion on compression. Learners could research which category of compression different file formats use (such as mp3, mp4). **(I)** Discuss transmission speeds for text, graphics and video and relate this (using the internet as the background) to the need for small file sizes, and particularly file compression. **(W)** **Resources:** Notes on compression techniques with exercises: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Sound_compression Nyquist theorem: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Nyquist-theorem Five pages of interesting explanation on compression. Useful for learners' research: http://computer.howstuffworks.com/file-compression2.htm Links to some interesting background reading for the more able learners: www.cs4fn.org/mathemagic/sonic.html

1.2 Communication and internet technologies

Syllabus ref	Learning objectives	Suggested teaching activities
1.2.1 Networks	• explain the client-server model of networked computers • give examples of applications which use the client-server model • describe what is meant by the World Wide Web (www) and the internet	Introduce the client-server model to learners. **(W)** In groups, provide learners with examples of client-server applications and ask them to discuss how they are used. **(G)** Get your learners to complete a 'match-terms-to-definition' exercise. **(P)** **Resources:** Brief notes on the internet and www: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Internet,_Intranet_and_World_Wide_Web#The_Internet Notes on the client-server model: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Client_server_model A comprehensive overview of the internet: http://en.wikipedia.org/wiki/Internet Information around internet access (building on the previous link): http://en.wikipedia.org/wiki/Internet_access A video of a teacher talking about the internet and the www (ignore intranet) (4:42 min): www.youtube.com/watch?v=KZNgyNPZEvw&list=PL997A0CD223D94B27 A video of a teacher talking about client-server and peer-peer network models (7:24 min): www.youtube.com/watch?v=AWFLGFV4R4c&list=PL997A0CD223D94B2

Syllabus ref	Learning objectives	Suggested teaching activities
	• explain how hardware is used to support the internet: networks, routers, gateways, servers • explain how communication systems are used to support the internet: The Public Switched Telephone Network (PSTN), dedicated lines, cell phone network • explain the benefits and drawbacks of using copper cable, fibre-optic cabling, radio waves, microwaves, satellites	Produce a diagram of a WAN using the hardware listed. Get your learners to research the role of each hardware item. **(I)** Get your learners to research different media (wired and wireless), some detail of the media itself (e.g. copper cable, fibre-optic cable, radio waves, microwaves, satellites) and some figures for transfer rates and ranges. Get them to produce a summary table. **(P)** **Resources:** A tutorial detailing network components: http://www.teach-ict.com/as_a2_ict_new/ocr/A2_G063/333_networks_coms/network_components/miniweb/index.htm A tutorial detailing optical and wireless technology: http://www.teach-ict.com/as_a2_ict_new/ocr/A2_G063/333_networks_coms/optical_wireless/miniweb/index.htm

Syllabus ref	Learning objectives	Suggested teaching activities
	• show understanding of bit streaming (both real-time and on-demand) • show understanding of the importance of bit rates/broadband speed on bit streaming	Get your learners to produce a summary of bit streaming and the impact of bit rates on bit streaming. Learners should put this into a practical context (e.g. watching videos over the internet). This could be completed as group work with each group of learners presenting their summary to the rest of the class. **(G)** **Resources:** Streaming media: Look at the section headed 'bandwidth and storage': http://en.wikipedia.org/wiki/Streaming_media

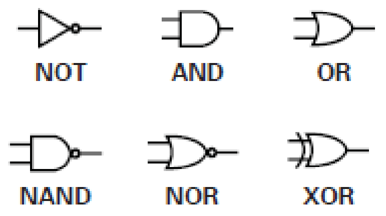
Syllabus ref	Learning objectives	Suggested teaching activities
1.2.2 IP addressing	• explain the format of an IP address and how an IP address is associated with a device on a network • explain the difference between a public IP address and a private IP address and the implication for security • explain how a Uniform Resource Locator (URL) is used to locate a resource on the World Wide Web (www) and the role of the Domain Name Service	Give learners a list of IP addresses of websites. Discuss what happens when they type in the IP address of the website rather than its URL **(I) (W)** Ask learners to explore why each part of an IP address is between 0 and 255. This can be used to revise the conversion of binary numbers to decimal. IPv6 topic could be used to revise conversion between hexadecimal and decimal. **(I)** Learners draw a diagram of what happens from when a user types in a URL until the web page is displayed in the browser. **(I)** **Resources:** Introductory notes for IP addresses and exercises: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/IP_addresses Notes on domain names and DNS: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Domain_names Notes on URL (ignore URI): http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/URIs

Syllabus ref	Learning objectives	Suggested teaching activities
1.2.3 Client- and server-side scripting	• describe the sequence of events executed by the client computer and web server when a web page consisting only of HTML tags is requested and displayed by a browser - Client-side ○ recognise and identify the purpose of some simple JavaScript code ○ describe the sequence of events executed by the client computer and web server when a web page with embedded client-side code is requested and displayed by a browser ○ show understanding of the typical use of client-side code in the design of an application - Server-side ○ recognise and identify the purpose of some simple PHP code ○ describe the sequence of events executed by the client computer and web server when a web page with embedded server-side code is requested and displayed by a browser ○ show understanding that an appropriately designed web application for accessing database data makes use of server-side scripting	Provide learners with simple examples of code (embedded java and embedded PHP) and discuss the different parts of the code and how to recognise these. This could be done in a practical way using a text editor and a browser. **(W)** Discuss the reasons why database data would be accessed using server-side scripting. **(W)** **Resources:** Short tutorials to set up a simple web page containing PHP code: http://php.net/manual/en/tutorial.firstpage.php www.w3schools.com/php/php_syntax.asp www.htmlgoodies.com/beyond/php/article.php/3472431/PHP-Tutorial-First-Page.htm An introduction to JavaScript: http://www.w3schools.com/js/js_intro.asp

1.3 Hardware

Syllabus ref	Learning objectives	Suggested teaching activities
1.3.1 Input, output and storage devices	• identify hardware devices used for input, output, secondary storage • show understanding of the basic internal operation of the following specific types of device: - keyboard - trackerball mouse - optical mouse - scanner - inkjet printer - laser printer - 3D printer - speakers - hard disk - solid state (flash) memory - optical discs - microphone - touchscreen • show understanding of the need for secondary (including removable) storage	Ask learners to produce a summary of the internal operation of hardware devices. This could be completed as group work where each group prepares a different type of device and presents to the rest of the class. **(G) (P)** **Resources:** Brief overview of hardware and software: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamentals_of_Computer_Systems/Hardware_and_software Brief overview of I/O devices: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Hardware_Devices/Input_and_output_devices Notes including keyboard, optical mouse, scanner: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Hardware_Devices/Input_devices, Notes on trackerball: http://en.wikipedia.org/wiki/Trackball Notes including inkjet printer, laser printer, speakers: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Hardware_Devices/Output_devices Notes including hard disk, optical disks, flash memory: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Hardware_Devices/Secondary_storage_devices

Syllabus ref	Learning objectives	Suggested teaching activities
1.3.2 Main memory	• show understanding of the need for secondary (including removable) storage - explain the differences between RAM and ROM memory - explain the differences between Static RAM (SRAM) and Dynamic RAM (DRAM)	Ask your learners to investigate which type of RAM chips their own/the school's computers have. **(P)** Get learners to complete a table of all the different types of main memory and their characteristics (speed, power consumption, relative cost of production). This could be produced as a poster for a classroom display. **(I)** Have a class discussion about why there are different types of main memory. **(W)** Get learners to complete a multiple-choice quiz. **(I)** **Resources:** Notes on how to check the type of memory in your computer: www.ehow.com/how_6467551_check-type-memory-computer-running.html Definition of computer memory: www.ehow.com/about_4675236_what-definition-computer-memory.html How RAM works: http://computer.howstuffworks.com/ram.htm Different types of RAM: http://computer.howstuffworks.com/ram3.htm The difference between static and dynamic RAM: http://computer.howstuffworks.com/question452.htm Links to ROM and RAM notes: http://computer.howstuffworks.com/computer-memory.htm Different types of RAM: www.ehow.com/list_6470557_different-types-ram-chips_.html

Syllabus ref	Learning objectives	Suggested teaching activities
1.3.3 Logic gates and logic circuits	• use the following logic gate symbols:  • understand and define the functions of NOT, AND, OR, NAND, NOR and XOR (EOR) gates including the binary output produced from all the possible binary inputs (all gates, except the NOT gate, will have two inputs only) • construct the truth table for each of the logic gates above • construct a logic circuit from either: - a problem statement, or - a logic expression • construct a truth table from either: - a logic circuit, or - a logic expression • show understanding that some circuits can be constructed with fewer gates to produce the same outputs	Give your learners a sheet with each of the five logic gates and associated truth table. Initially the output columns are empty. Explain why the NOT truth table has only two possible inputs whilst the other truth tables have four possible combinations of input. **(W)** Ask your learners complete the output columns for each of the truth tables with appropriate explanations. **(I)(F)** Give your learners a worksheet with a number of examples of more complex logic circuits each of which is comprised of a number of logic gates. Show learners how to tackle a couple of the problems. If the logic circuit has three inputs explain why there are eight possible input combinations in the truth table. Show learners that labelling intermediate parts of the circuit and including them in the table is often of assistance in completing the truth table. **(W)** Get learners to complete the other problems. **(I) (F)** Give your learners a worksheet with a number of examples of written statements that can be turned into simple logic circuits. Show learners how to tackle a couple of the problems. **(W)** Get your learners to complete the other problems. **(I)(F)** Learners can check their answers and experiment with circuits using a logic gate simulator. (There are different versions on the web. Check which one suits you best.) **Resources:** Notes of logic gates and simple exercises: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Logic_Gates Exercises on simple gate combinations: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Boolean_gate_combinations Exercises on building circuits: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Building_circuits

Syllabus ref	Learning objectives	Suggested teaching activities
		Logic gate simulator: www.kolls.net/gatesim/ A set of three videos talking about binary logic: *Part 1: Basics* www.youtube.com/watch?v=_76g8EM4DVU&list=PL997A0CD223D94B27 (9:26 min) *Part 2: Advanced* www.youtube.com/watch?v=jaPGb3OwRkA&list=PL997A0CD223D94B27 (10:43 min) *Part 3: Algebra* www.youtube.com/watch?v=YsaHu2_VfGk&list=PL997A0CD223D94B27 (8:06 min)

1.4 Processor fundamentals

Syllabus ref	Learning objectives	Suggested teaching activities
1.4.1 CPU architecture	• show understanding of the basic Von Neumann model for a computer system and the stored program concept • show understanding of the roles carried out by registers, including the difference between general purpose and special purpose registers: Program Counter, Memory Data Register, Memory Address Register, Index Register, Current Instruction Register and Status Register • show understanding of the roles carried out by the Arithmetic and Logic Unit (ALU), Control Unit and system clock • show understanding of how data are transferred between various components of the computer system using the address bus, data bus and control bus • show understanding of how the bus width and clock speed are factors that contribute to the performance of the computer system • show understanding of the need for ports to provide the connection to peripheral devices	Give and then test your learners' basic understanding of the three primary elements of the CPU, covering (briefly) the functions of each element. Reinforce this element orally, via worksheets or using a computer simulation **(W)**. Introduce the concept of Von Neumann architecture – any computer that takes a single instruction then obeys it before processing the next instruction. Describe the contents and the use of the following registers: Sequence Control Register (Program Counter) Current Instruction Register Memory Address Register Memory Buffer Register Using a simulation, such as Little Man Computer, demonstrate how and when each register is used in the cycle. Get your learners to type in a simple program and use the 'step' feature to see this. **(I)** **Resources:** Five pages explaining the theory of Von Neumann architecture: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_3/vonn_neuman/miniweb/index.htm Notes and exercises on computer architecture: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Machine_Level_Architecture/Internal_and_external_hardware_components_of_a_computer Notes and exercises on stored program concept: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet

Syllabus ref	Learning objectives	Suggested teaching activities
		Notes and exercises on parts of the processor: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Machine_Level_Architecture/Structure_and_role_of_the_processor

Syllabus ref	Learning objectives	Suggested teaching activities
1.4.2 The fetch−execute cycle	• describe the stages of the fetch−execute cycle • show understanding of 'register transfer' notation • describe how interrupts are handled	Prepare a diagram showing the flow of data/instructions through the registers. Include the use of Data/Address/Control buses. (Make it clear what is being transferred on the buses: data/instructions on data bus; addresses on address bus; signals on control bus.) If possible provide a demonstration of the fetch−execute cycle: (www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_3/fetch_execute_cycle/miniweb/index.htm) Using a set of simple Assembly Language/Machine Code instructions trace the contents of each of the registers, this can be done as a whole class exercise giving the opportunity to work through the cycle several times using different types of instruction. This could again be all done using the Little Man Computer simulation **Resources:** Theory notes and a presentation on the fetch−execute cycle: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_3/fetch_execute_cycle/theory_fetch_execute.html Notes and exercises on register transfer notation: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Machine_Level_Architecture/The_Fetch%E2%80%93Execute_cycle_and_the_role_of_registers_within_it Theory notes on the fetch−execute cycle: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_1/interrupts/miniweb/pg4.htm Little Man Computer simulation: http://peterhigginson.co.uk/LMC/

Syllabus ref	Learning objectives	Suggested teaching activities
1.4.3 The processor's instruction set	• show understanding that the set of instructions are grouped into instructions for: - data movement (register to main memory and vice versa) - input and output of data - arithmetic operations - unconditional and conditional jump instructions - compare instructions - modes of addressing: immediate, direct, indirect, indexed, relative (No particular instruction set will be expected but candidates should be familiar with the type of instructions given in the table in the syllabus on page 20.)	Get learners to write simple programs in assembly code such as: read a number from memory add another number from another memory location store the result in a third memory location Use a simulator, so learners can check their programs. **(I)(F)** Stepping through the execution slowly so learners can follow what happens inside the different registers/memory locations. Learners should be able to predict what the next step will be. This could be done as a whole class exercise if the whole class can see the same output screen. **(W)** **Resources:** Notes and exercises on instruction set: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Machine_Level_Architecture/Machine_code_and_processor_instruction_set Notes on the different addressing modes: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_8/lowlevel/miniweb/index.htm Further exercises: https://sites.google.com/a/bxs.org.uk/mrkershaw/lmc

Syllabus ref	Learning objectives	Suggested teaching activities
1.4.4 Assembly language	• show understanding of the relationship between assembly language and machine code, including symbolic and absolute addressing, directives and macros • describe the different stages of the assembly process for a 'two-pass' assembler for a given simple assembly language program • trace a given simple assembly language program	Show an assembly language program, highlighting the assembly language statement syntax: <optional label><opcode mnemonic><operand> **(W)** Show the translated version to highlight the one-to-one connection between the two forms of the instruction. **(W)** If the assembler shows evidence of two passes and the use of a symbol table then use these in explaining the assembly process. Also ensure that the explanation of how the opcode mnemonic is converted using an opcode table. To complete the picture mention/show directives and macros. **(W)** **Resources:** Definition of assembly language: www.webopedia.com/TERM/A/assembly_language.html Definition of machine language: www.webopedia.com/TERM/M/machine_language.html Links to theory notes on low-level languages including a worked example: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_8/features/miniweb/index.htm

1.5 System software

Syllabus ref	Learning objectives	Suggested teaching activities
1.5.1 Operating system	• describe why a computer system requires an operating system • explain the key management tasks carried out by the operating system	Discuss with learners what an operating system is. **(W)** If possible show examples of different operating systems. Ask learners in groups to identify, from their experiences, what they think all operating systems can do. **(G)** **Resources:** Series of pages describing OS tasks: http://computer.howstuffworks.com/5-important-operating-system-jobs.htm#page=1 Series of pages of how and OS works: http://computer.howstuffworks.com/operating-system1.htm
1.5.2 Utility programs	• show an understanding of the need for typical utility software used by a PC computer system: - disk formatter - virus checker - defragmenter software - disk contents analysis/disk repair software - file compression - backup software	Ask learners to rapidly list some utility software they may have used or installed. Draw up a summary table (utility, what it does) based on learner contributions. It may help the discussion to classify the utility as either: configuring, optimising, or maintaining the system. **(W)** **Resources:** Notes and exercises on system software: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamentals_of_Computer_Systems/System_software#Utility_programs Utility software examples: http://study.com/academy/lesson/systems-software-utility-software-device-drivers-firmware-gui.html

Syllabus ref	Learning objectives	Suggested teaching activities
1.5.3 Library programs	• show an understanding that software under development is often constructed using existing code from program libraries • describe the benefits to the developer of software constructed using library files, including Dynamic Link Library (DLL) files • draw on experience of the writing of programs which include library routines	This topic could be combined with Section 2. Learners research the libraries available for their programming language. How does a programmer use programs from such libraries? **(I)** **Resources:** Notes on library programs: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamentals_of_Computer_Systems/System_software#Library_programs

Syllabus ref	Learning objectives	Suggested teaching activities
1.5.4 Language translators	• show an understanding of the need for: assembler software for the translation of an assembly language program a compiler for the translation of a high-level language program an interpreter for execution of a high-level language program • explain the benefits and drawbacks of using either a compiler or interpreter • show awareness that high-level language programs may be partially compiled and partially interpreted, such as Java	Initially, demonstrate the use of a compiler and the use of an interpreter. **(W)** Highlight the differences between compilation and interpretation including, at a minimum that: **compiler** translates the whole program (source code) into object code that can be stored and re-used **interpreter** translates and executes a program line by line. No object code is stored for further use – a program has to be translated each time it is used. Discuss the advantages and disadvantages of compilation and interpretation highlighting when it would be appropriate to use a compiler or an interpreter (e.g. use an interpreter during program development as errors can be easily checked and modified). As learners have used translators they should be able to contribute to a discussion. Create a table with some statements about compilers and some statements about interpreters. Ask your learners to read the statements and tick which apply to a compiler and which apply to an interpreter. **(I)(F)** **Resources:** Link to theory notes on compilers and interpreters: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_2/translators_compilers/miniweb/pg14.htm Short notes and exercises on program translators: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamentals_of_Computer_Systems/Types_of_program_translator

1.6 Security, privacy and data integrity

Syllabus ref	Learning objectives	Suggested teaching activities
1.6.1 Data security	• explain the difference between the terms security, privacy and integrity of data • show appreciation of the need for both the security of data and the security of the computer system • describe security measures designed to protect computer systems, ranging from the stand-alone PC to a network of computers, including: - user accounts - firewalls - general authentication techniques, including the use of password and digital signatures • describe security measures designed to protect the security of data, including: - data backup - a disk-mirroring strategy - encryption - access rights to data (authorisation) • show awareness of what kind of errors can occur and what can be done about them	Discuss the problems of ensuring the confidentiality of data as it is being transferred across and stored at nodes on an open network, where coding and transmission methods are freely available. Include the following ideas in your discussion: prevention of access to data when stored (e.g. physical security, use of access levels and passwords) protection of data, from malicious interference, during transmission (e.g. use of encryption, screening of cables, problems with radio transmission, benefits of packet switching etc.) use of authorisation techniques to ensure that confidential information only reaches the intended recipient (e.g. use of passwords, responses to special questions, provision of memorable data etc.) **(W)** Ask learners to read a case study based on a scenario about security, privacy and integrity of data issues, and then complete some questions. **(I)(F)** **Resources:** Comments on encryption: www.howstuffworks.com/encryption.htm Comments on authentication methods: http://computer.howstuffworks.com/computer-user-authentication-channel.htm

Syllabus ref	Learning objectives	Suggested teaching activities
1.6.2 Data integrity	• describe error detection and correction measures designed to protect the integrity of data, including: - data validation - data verification for data entry - data verification during data transfer, including: ○ parity check ○ checksum check	Discuss the need for the accurate input of data and the ways in which we can check that the data is correct. Make learners aware of the fact that data can be checked both automatically and manually. Ask them to suggest the limitations of both methods. **(W)** Will a computer **know** there is a mistake if a date of birth is typed in as 16/12/85? How about 16/13/85? Describe the meaning of the term valid and emphasise the fact that a computer can only check for **valid** data. Look at checks for existence, range, character, format, length and check digit (in the case of barcodes etc.) as automated on data entry. Discuss what verification means. Describe verification of data as manual checking that the data has been typed in correctly, sometimes visually but more often by double data entry. Get learners to complete a matching terms and definitions activity for validation and verification methods. They match the term to the definition and then put them in the validation or verification pile depending on what they are. **(I)(F)** Discuss the need for parity checks and checksums as well as other data checking systems at this point. Include notes on echoing back – to include the need for Duplex or Half-Duplex to allow this to happen. **(W)** The learners should be able to calculate parity bits (both odd and even should be understood). Give learners a table of binary numbers and ask learners to calculate the parity bit for each depending it is odd or even. Learners could also be given numbers where the arity bit is already calculated and asked to identify if it is correct or not. **(I)(F)** **Resources:** Notes on validation and verification: http://www.bbc.co.uk/education/guides/zdvrd2p/revision Introduction to error checking: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Error_checking_and_correction

Syllabus ref	Learning objectives	Suggested teaching activities
		Notes and exercises for parity checks: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Parity_bits Notes on checksum (individual algorithms not required): http://en.wikipedia.org/wiki/Checksum Class activity on error detection and correction: http://csunplugged.org/error-detection

1.7 Ethics and ownership

Syllabus ref	Learning objectives	Suggested teaching activities
1.7.1 Ethics and the computing professional	• show a basic understanding of ethics • explain how ethics may impact on the job role of the computing professional • show understanding of the eight categories listed in the ACM/IEEE Software Engineering Code of Ethics • demonstrate the relevance of these categories to some typical software developer workplace scenarios • show understanding of the need for a professional code of conduct for a computer system developer	Give learners a variety of different scenarios and let them discuss the ethics of the situation. **(G)** **Resources:** The eight categories of software engineering code of ethics: www.sqa.org.uk/e-learning/ProfIssues03CD/page_04.htm British Computer Society code of conduct: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Consequences_of_Uses_of_Computing/Code_of_conduct

Syllabus ref	Learning objectives	Suggested teaching activities
1.7.2 Ownership of software and data	• show understanding of the concept of ownership and copyright of software and data • describe the need for legislation to protect ownership, usage and copyright • discuss measures to restrict access to data made available through the internet and World Wide Web • show understanding of the implications of different types of software licensing: Free Software Foundation, the Open Source Initiative, shareware and commercial software	Get your learners to summarise the different types of licensing in a table. **(I)(F)** Discussion of the reasons of copyright using a role play scenario. **(W)(G)** **Resources:** Copyright: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Consequences_of_Uses_of_Computing/Legislation#Copyright Cases of hacking: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Consequences_of_Uses_of_Computing/Hacking Free software foundation: www.fsf.org/about/ Open source initiative: http://opensource.org/osd Notes on shareware: http://en.wikipedia.org/wiki/Shareware Notes on commercial and free and open-source software: http://en.wikipedia.org/wiki/Commercial_software

1.8 Database and data modelling

Syllabus ref	Learning objectives	Suggested teaching activities
1.8.1 Database Management Systems (DBMS)	• show understanding of the limitations of using a file-based approach for the storage and retrieval of large volumes of data • describe the features of a relational database which address the limitations of a file-based approach • show understanding of the features provided by a DBMS to address the issues of: - data management, including maintaining a data dictionary - data modelling - logical schema - data integrity - data security, including backup procedures and the use of access rights to individuals/groups of users • show understanding of how software tools found within a DBMS are used in practice: - developer interface - query processor • show awareness that high-level languages provide accessing facilities for data stored in a database	Introduce the advantages of using a relational database rather than a flat file including: **(W)** • data independence • data consistency • lack of duplication of data • less redundant data Get learners to provide examples of everyday databases that they may use (e.g. phone contacts, membership of a sports club). **(I)** **Resources:** Definition of relational database: http://computer.howstuffworks.com/question599.htm Theory notes on databases: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_9/database_design/miniweb/index.htm Theory notes on concepts: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_9/ddl/miniweb/index.htm

Syllabus ref	Learning objectives	Suggested teaching activities
1.8.2 Relational database modelling	• show understanding of, and use, the terminology associated with a relational database model: entity, table, tuple, attribute, primary key, candidate key, foreign key, relationship, referential integrity, secondary key and indexing • produce a relational design from a given description of a system • use an entity-relationship diagram to document a database design	Using a practical example of a previously set up relational database introduce the concepts of: • tables • primary keys • foreign keys • secondary keys • views of data Demonstrate and explain the purpose of each of these concepts using the pre-prepared database and introduce the learners to the formally set out underlying data structures. **(W)** For example, TableLoan (<u>LoanNo</u>, BookNo, LibMemNo, BorrowDate, ExpReturnDate, ActReturnDate). Where LoanNo is the primary key of the loan table, BookNo and LibMemNo are foreign keys from other tables in a library database. Introduce the concepts of entities and relationships (one-to-one, one-to-many, many-to-many). Use the board to draw and illustrate relationships. **(W)** Use everyday occurrences to demonstrate these concepts (e.g. the learner−teacher model can be discussed showing the idea of a many-to-many relationship between learner and teacher and how the introduction of other entities such as class meeting can help organise the model). Explain how the relationships need to be carefully labelled in order to show understanding. Similar data structures can be used to the ones prepared for the normalisation exercise, this will help enforce how these two techniques complement each other. **(W)** Give learners a worksheet with some entities and ask them to identify the correct relationships between each. **(I)(F)** **Resources:** Introductory notes on databases: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/Databases

Syllabus ref	Learning objectives	Suggested teaching activities
		Notes on database keys: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/Primary_keys Link to theory notes on database modelling: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_9/er_diagrams/miniweb/index.htm Notes on database keys: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_9/dbkey/miniweb/index.htm

Syllabus ref	Learning objectives	Suggested teaching activities
	• show understanding of the normalisation process: First (1NF), Second (2NF) and Third Normal Form (3NF) • explain why a given set of database tables are, or are not, in 3NF • make the changes to a given set of tables which are not in 3NF to produce a solution in 3NF, and justify the changes made	Demonstrate the principles of normalisation starting with a flat file data structure and working through the stages of normalisation: • 1NF – remove repeating data • 2NF – remove partial key dependencies • 3NF – remove non-key dependencies Choose your examples very carefully to ensure the example used for demonstration and the first few that the learners attempt need work to be done at all stages (many examples may not yield composite keys so there can be no partial key dependencies). **(W)** Provide pre-determined scenarios (e.g. customer orders, student records etc.) that allow the learners to identify, specify and normalise the data structures required. **(I)(F)** **Resources:** Normalisation: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/Entity_relationship_modelling Theory on normalisation: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_9/normalisation/miniweb/index.htm

Syllabus ref	Learning objectives	Suggested teaching activities
1.8.3 Data Definition Language (DDL) and Data Manipulation Language (DML)	• show understanding that DBMS software carries out: - all creation/modification of the database structure using its DDL - query and maintenance of data using its DML • show understanding that the industry standard for both DDL and DML is Structured Query Language (SQL) - show understanding of a given SQL script - write simple SQL (DDL) commands using a sub-set of commands for: ○ creating a database (CREATE DATABASE) ○ creating a table definition (CREATE TABLE) ○ changing a table definition (ALTER TABLE) ○ adding a primary key or foreign key to a table (ADD PRIMARY KEY)	Introduce the main functions of a DBMS: • Data Dictionary (an internal file containing the name, description, characteristics, relationships for each data item and information about programs and users. • Data Description/Definition Language (DDL) • Data Manipulation Language (DML) Explain that this information is stored with the data in a database system. Learners may have used a GUI to define and manipulate data but a demonstration of the underlying commands actually used (e.g. showing the SQL commands produced by a QBE query) could be used to show the functions of a DDL and a DML as SQL has both properties. **(W)** Give learners a paragraph about DBMS software that has some gaps that need filling. As them to complete the gaps. **(I) (F)** Using software, such as MS Access, allow users to create simple SQP statements to set up a database. **(I)** **Resources:** Structured Query Language (SQL): http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/SQL SQL tutorials: http://sqlzoo.net/wiki/Main_Page Data definition language (DDL): http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/Data_definition_language http://en.wikipedia.org/wiki/Data_definition_language

Syllabus ref	Learning objectives	Suggested teaching activities
	- write a SQL script for querying or modifying data (DML) which are stored in (at most two) database tables ○ Queries: ○ SELECT, FROM, WHERE, ORDER BY, GROUP BY, INNER JOIN - Data maintenance: • INSERT INTO, DELETE FROM, UPDATE	This topic is best delivered in a practical way, so learners can check their answers. Use a practical example of a previously set up relational database with sufficient data to write queries that give groups of records as results. Software, such as MS Access, allows switching between SQL view and Design view to allow learners to work from the familiar to the new. **(I)** **Resources:** SELECT: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/SELECT INSERT: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/INSERT DELETE: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/DELETE UPDATE: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/UPDATE Data manipulation language (DML): http://en.wikipedia.org/wiki/Data_manipulation_language

2.1 Algorithm design and problem-solving

Syllabus ref	Learning objectives	Suggested teaching activities
2.1.1 Algorithms	• show understanding that an algorithm is a solution to a problem expressed as a sequence of defined steps • use suitable identifier names for the representation of data used by a problem - summarise identifier names using an identifier table • show understanding that many algorithms are expressed using the four basic constructs of assignment, sequence, selection and repetition • show understanding that simple algorithms consist of input, process, output at various stages • document a simple algorithm using: - structured English - pseudocode (on the examination paper, any given pseudocode will be presented using the Courier New font) - program flowchart • derive pseudocode or a program flowchart from a structured English description of the problem	Produce the algorithm for making a cup of tea (or coffee). Remind learners about how to draw flowcharts and ask them to attempt to draw a flowchart to show how to make a cup of tea. This will lead to discussions about selection, sequence and repetition. **(W)** **Examples (taken from how to make a cup of tea)** *Sequence* Add water to kettle; Put kettle on heat source (as example of sequence) *Decisions/selection with Y/N solutions* Do you take sugar? Discuss framing the questions to always give Yes or No answers. Create a flowchart to illustrate these steps. *Selection: IF…Then…Else constructs* Do you take sugar? If Yes then go to section which adds sugar to the cup, if No go to the section for milk. Create a further flowchart for this section (perhaps as a module called Sugar). **(W)** *Iteration – use from cup of tea in the sugar module* Add a little sugar – Is this enough? If Yes return from the module If not go back to 'Add a little sugar'.

Syllabus ref	Learning objectives	Suggested teaching activities
	• derive pseudocode from a given program flowchart or vice versa	Summarise that sequence, selection and iteration form the three basic programming constructs. **(W)** Show learners some examples of program flowcharts and pseudocode. Do dry runs on the examples to show learners i. how to tackle dry runs and ii. how to interpret flowcharts symbols and pseudocode vocabulary. **(W)** Give learners guidance on the symbols to be used in producing flowcharts and the words to be used in the pseudocode. Get leaners to produce flowcharts and pseudocode for a number of simple problems. Give the learners some further questions on dry running some algorithms and also some questions on producing their own flowcharts and pseudocode. Show model solutions to the question. **(I) (F)** **Resources:** Notes on algorithm design: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Algorithm_design Notes on pseudocode: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Pseudo_code *RAPTOR*, free program flowchart interpreter software that allows learners to draw a flowchart and check its functioning by executing it: http://raptor.martincarlisle.com/ Gliffy is a flowchart creation software: https://www.gliffy.com/

Syllabus ref	Learning objectives	Suggested teaching activities
	• use the process of stepwise refinement to express an algorithm to a level of detail from which the task may be programmed • decompose a problem into sub-tasks leading to the concept of a program module (procedure/function) • show an appreciation of why logic statements are used to define parts of an algorithm solution • use logic statements to define parts of an algorithm solution	Discuss how to find the area of a 'house' made up from a square and a triangle by working out the area of the triangle, working out the area of the square and then adding the two together. **(W)** Use this to explain what a 'top down' approach is – a large complex problem broken into smaller more manageable pieces. When each of the smaller problems have been solved then all the pieces are put together to give an overall solution. **(W)** Introduce concept of modularity. More than one person or team of people can be engaged in solving different parts of the same problem at the same time. Therefore the problem can be solved more quickly. **(W)** Give a similar problem to four 'teams' in the classroom. The problem is to design a new computerised traffic light system for (name a local set of highway traffic lights controlling a road junction). Identify the four areas to be addressed as discussed in the production line example. **(G)** Give each group time to think of possible solutions, put all solutions together and see if that fulfils the original task. In this instance it does not matter if the group's solutions work – if not it is better to provoke discussion about definition of each group's task, what we asked them to do, what input they required and what output they were expected to give. This should develop the idea of modular notation (on input, process, on output) as used in standard programming techniques. **(G)(F)** **Resources:** Stages of problem solving: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Stages_of_problem_solving Step-wise refinement: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Top-down_design_/_Step-wise_refinement Structured programming: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Fundamentals_of_Structured_Programming

Syllabus ref	Learning objectives	Suggested teaching activities
2.1.2 Structure chart	• use a structure chart to express the inputs into and output from the various modules/procedures/functions which are part of the algorithm design • describe the purpose of a structure chart • construct a structure chart for a given problem • derive equivalent pseudocode from a structure chart	Show how a tree-like diagram can illustrate the stepwise refinement that is the outcome of a 'top down' approach. Discuss the need to capture repetition and selection in a structure diagram and how this can be achieved. **(W)** Give the learners some exercises to produce structure diagrams for simple problems. **(I) (F)** Give learners some structure diagrams from which to produce pseudocode. **(I) (F)** **Resources:** Notes on structure charts: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Structure_charts

Syllabus ref	Learning objectives	Suggested teaching activities
2.1.3 Corrective maintenance	• perform white-box testing by: - selecting suitable data - using a trace table • identify any error(s) in the algorithm by using the completed trace table • amend the algorithm if required	Demonstrate the use of dry runs (desk checking) on simple arithmetic programs with loops. **(W)** Start with algorithms/programs without errors. Then give learners algorithms/programs with a simple error that they can find as a result of doing a dry-run. They should then be able to correct the error and check the revised algorithm/program works correctly. **(I) (F)** **Resources:** Notes and exercises on trace tables: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Trace_tables

Syllabus ref	Learning objectives	Suggested teaching activities

2.1.4 Adaptive maintenance	• make amendments to an algorithm and data structure in response to specification changes
	• analyse an existing program and make amendments to enhance functionality
		Give learners an algorithm/program they can amend. **(P) (F)**
		
		For example:
		A program that reads in 20 numbers using a FOR loop could be amended so it reads in numbers until some terminal value.
		
		The following bubble sort algorithm could be improved:
		
		`FOR value1 ← 1 to (n-1)`
		`   FOR value2 ← 1 to (n-1)`
		`      COMPARE List[value1] with List[value2]`
		`      IF greater THEN swap elements`
		`   ENDFOR`
		`ENDFOR`

2.2 Data representation

Syllabus ref	Learning objectives	Suggested teaching activities
2.2.1 Data types	• select appropriate data types for a problem solution • use in practical programming the data types that are common to procedural high-level languages: integer, real, char, string, Boolean, date (pseudocode will use the following data types: INTEGER, REAL, CHAR, STRING, BOOLEAN, DATE, ARRAY, FILE) • show understanding of how character and string data are represented by software including the ASCII and Unicode character sets	Explain the features of, and difference between, different data types. Identify suitable data for different functions. Explain which data types are suitable for different data. Explain relative storage sizes of different data types. **(W)** Give learners a worksheet to select the correct data types for different samples of data. Enhance this to include storage sizes. Marking these worksheets orally in class should provoke and stimulate discussion on different storage types and the relative merits of each for specific functions. Ensure that all data types listed are covered. **(I) (F)** Ask learners to write a simple program that reads in a character and outputs the ASCII/Unicode value. Variations of this could be to output the next/previous letter in the alphabet by adding/subtracting from the ASCII value and converting back into a character. **(I)** Let learners explore the difference between for example, the number 2 and the character 2 by outputting the ASCII value of a digit. **Resources:** Notes on data types: http://en.wikipedia.org/wiki/Data_type Notes on built-in data types: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Built-in_data_types Constants and variables: www.pp4s.co.uk/main/tu-ucv-using-constants-and-variables-intro.html

2.2.2 Arrays	• use the technical terms associated with arrays including upper and lower bound • select a suitable data structure (1D or 2D array) to use for a given task • use pseudocode for 1D and 2D arrays (pseudocode will use square brackets to contain the array subscript, for example a 1D array as A[1:n] and a 2D array as C[1:m, 1:n]) • write program code using 1D and 2D arrays • write algorithms/program code to process array data including: sorting using a bubble sort searching using a linear search.	Demonstrate the purpose of an array using an example. **(W)** Explain the purpose and structure of one-dimensional arrays. Explain memory allocation, initialising arrays and reading data into arrays. **(W)** Set worksheet exercises to practise setting up one-dimensional arrays and reading data into these arrays. **(I) (F)** As a class activity or in small groups – design and write routine/s to perform a simple serial search on an array. **(G)** Use a further example to demonstrate the need for multi-dimensional arrays and give learners similar exercises to work on one-dimensional arrays. Discuss the need for dimensioning arrays and demonstrate how to do this. **(W)** Introduce the concept of bubble sort / linear search and let learners use cards with different numbers and manually work through the process. **(W)** Learners attempt to write the algorithm from memory of previous exercise. **(I)** **Resources:** 1D arrays: www.pp4s.co.uk/main/tu-arrays-intro.html 1D arrays: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/One-Dimensional_Arrays 2D arrays: www.pp4s.co.uk/main/tu-arrays-2D-arrays.html 2D arrays: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Two-Dimensional_Arrays

2.2.3 Files	• show understanding of why files are needed
	• use pseudocode for file handling:
	
	`OPENFILE <filename> FOR READ/WRITE/APPEND // Open file (understand the difference between various file modes)`
	`READFILE <filename>,<string> // Read a line of text from the fi le`
	`WRITEFILE <filename>,<string> // Write a line of text to the fi le`
	`CLOSEFILE // Close file`
	`EOF() // function to test for the end of the file`
	
	• write program code for simple file handling of a text file, consisting of several lines of text
		Discuss how records in a sequential file can be stored by opening a file, writing a record and then closing the file. **(W)**
		
		Discuss how a sequential file can be searched for a particular record and its contents output. **(W)**
		
		Show how the algorithms produced above can be implemented in a program that: opens a file initially and closes it at the end. A user can choose, via a menu, to read a chosen record, update a chosen record, insert a new record and append a new record. Discuss the syntax of the file operation statements to clarify how they are achieved using the particular procedural language. It may be beneficial, if possible, to look at the file records before and after a number of operations have been carried out on the file. This should help learners to understand more clearly the file operations that are carried out but also how the records are actually stored. **(W)**
		
		Resources:
		
		File handling:
		http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/File_handling
		
		Text file processing:
		www.pp4s.co.uk/main/tu-stringman-file-intro.html
		
		Sequential files:
		www.dreamincode.net/forums/topic/29575-file-handling-in-visual-basic-6-part-1-sequential-files
		
		Python file handling:
		www.pythonforbeginners.com/cheatsheet/python-file-handling/

2.3 Programming

Syllabus ref	Learning objectives	Suggested teaching activities

2.3.1 Programming basics	• write a program in a high-level language
	• implement and write a program from a given design presented as either a program flowchart or pseudocode
	• write program statements for:
	- the declaration of variables and constants
	- the assignment of values to variables and constants
	- expressions involving any of the arithmetic or logical operators
	given pseudocode will use the following structures:
	`DECLARE <identifier> : <data type> // declaration`
	`<identifier> ← <value> or <expression> // assignment)`
	`INPUT <identifier>`
	`OUTPUT <string>`
	`OUTPUT <identifier(s)>`
		Give out a printed copy of a short program that uses both variables and constants. Discuss briefly the terms variable, constant, identifier and reserved word/keyword. **(W)**
		
		Get learners to list all the variables, constants, identifiers and reserved words/keywords present in the program. Check answers ensuring that the terms have been correctly understood. **(I)**
		
		Discuss with learners the fact that some languages require variables to be declared before use whilst other languages do not. Discuss why it is useful to declare and name a constant. Show an example of code that demonstrates variable and constant declarations in both the ‘main’ program and in subroutines/procedures/functions. Use this code to discuss scope with learners and also the advantages of declaring constants. **(W)**
		
		Show learners a program for finding the average of a set of numbers (the number of numbers is input by the user) but written with ‘unsuitable’/‘obscure’ variable names. Ask learners to comment on the program code. When the idea of using meaningful identifier names has been grasped get learners to rewrite the program changing the identifier names. **(I)**
		
		Discuss techniques for naming identifiers that aid readability (e.g. use of space, underscore, and capital letters). Often there are conventions about the names that are used. **(W)**
		
		Discuss the benefits of initialising variables. Give some examples where uninitialised variables could lead to either run-time errors or erroneous results. Show that an uninitialised variable has a value but not a predictable one. **(W)**
		
		Discuss the advantages of putting comments into code. Show, using examples, that too many comments can be as ineffective as too few comments. **(W)**
		
		Demonstrate with examples the differences in making sense of code structure when indentation and formatting are used. **(W)** Discuss the nature of an assignment statement: an expression is evaluated and its result is assigned to a variable.

Syllabus ref	Learning objectives	Suggested teaching activities
2.3.2 Transferable skills	• recognise the basic control structures in a high-level language other than the one chosen to be studied in depth • appreciate that program coding is a transferable skill	Provide learners with programs written in a different programming language. For example, if the chosen programming language is VB, give a program written in Pascal. Ask learners to translate the program in the chosen programming language. The result should be tested to see if it produces the correct output. **(I)** **Resources:** Commenting programs: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Comments Inputs and outputs in programming: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Input_and_output Operators: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Arithmetic_operators www.pp4s.co.uk/main/tu-op-intro.html Defining constants: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Constant_Definitions Naming conventions: http://en.wikipedia.org/wiki/Naming_convention_%28programming%29

Syllabus ref	Learning objectives	Suggested teaching activities

2.3.3 Selection	• use an 'IF' structure including the 'ELSE' clause and nested IF statements
	- given pseudocode will use the following structure:
	`IF <condition>`
	`  THEN`
	`    <statement(s)>`
	`  ENDIF`
	- or, including an 'else' clause:
	`IF <condition>`
	`  THEN`
	`    <statement(s)>`
	`  ELSE`
	`    <statement(s)>`
	`ENDIF`
	• use a 'CASE' structure
	- given pseudocode will use the following structure:
	`CASE OF <identifier>`
	`  <value 1>: <statement>`
	`  <value 2>: <Statement>`
	`  ...`
	`ENDCASE`
	• alternatively:
	`CASE OF <identifier>`
	`  <value 1>: <statement>`
	`  <value 2>: <Statement>`
	`  ...`
	`  OTHERWISE <statement>`
	`ENDCASE`
		Demonstrate the use of IF and CASE statements using both pseudocode and programming language examples. Stress when is it appropriate to use each − although we can use the IF statement for very complex (nested) condition testing, the CASE statement usually makes it easier to read the code. **(W)**
		
		Provide learners with some code examples that include CASE and IF selections. Some of these examples should use them correctly, some should not. In pairs, learners should identify those that do use them correctly and those that do not. They should re-write the codes that do not. For example, provide a program where CASE has been used but and IF statement would be more efficient, and vice-versa. **(P)**

Syllabus ref	Learning objectives	Suggested teaching activities

2.3.4 Iteration	• use a 'count controlled' loop:
	- given pseudocode will use the following structure:
	`FOR <identifier> ← <value1> TO <value2>`
	`  <statement(s)>`
	`ENDFOR`
	- alternatively:
	`FOR <identifier> ← <value1> TO <value2> STEP <value3>`
	`  <statement(s)>`
	`ENDFOR`
	• use a 'post-condition' loop:
	- given pseudocode will use the following structure:
	`REPEAT`
	`  <statement(s)>`
	`UNTIL <condition>`
	• use a 'pre-condition' loop
	- given pseudocode will use the following structure:
	`WHILE <condition>`
	`  <statement(s)>`
	`ENDWHILE`
	• justify why one loop structure may be better suited to a problem than the others
		Use examples to demonstrate the different types of iteration: number of iterations known initially (use of FOR-NEXT statements) and number of iterations not known initially (use of REPEAT-UNTIL or WHILE-ENDWHILE). **(W)**
		
		Explain the need for WHILE-ENDWHILE (e.g. reading records from a file that might contain zero records). **(W)**
		
		Provide learners with some code examples and ask them to identify if the most efficient loop has been used in each case. In pairs, they should rewrite those that are not the most efficient. For example, correct a counting loop to be a condition loop if that were a more efficient solution. **(P)**
		
		Resources:
		Control flow:
		http://en.wikipedia.org/wiki/Control_flow
		
		Introduction to selection:
		www.pp4s.co.uk/main/tu-selection-intro.html
		
		IF and CASE statements:
		www.delphibasics.co.uk/Article.asp?Name=Logic
		http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Selection
		
		Introduction to iteration:
		www.pp4s.co.uk/main/tu-iteration-intro.html
		
		Counting and condition loops:
		http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Iteration

Syllabus ref	Learning objectives	Suggested teaching activities
2.3.5 Built-in functions	• use a subset of the built-in functions and library routines supported by the chosen programming language. This should include those used for: - string/character manipulation - formatting of numbers - random number generator • use the information provided in technical documentation describing functions/procedures	Discuss that arithmetic operations cannot be performed on strings. Explain that there are other operations that are useful for manipulating strings and that these are usually in the form of the functions (which need parameter(s) and return results). **(W)** Issue a hand-out that has the name of the function, description of what the function does, and an illustrative example. Look at the concatenation and comparison of two strings and show how these operations are expressed. Give learners examples to answer. **(I)** Remind learners that the ASCII and CHAR functions reinforce the idea that strings are actually stored as a series of (binary) numbers and that comparison of characters is actually the comparison of their (binary) codes. Consequently, it is possible to perform a comparison between '2' and 'a' and get a valid result.

Syllabus ref	Learning objectives	Suggested teaching activities

2.3.6 Structured programming	• use a procedure
	• explain where in the construction of an algorithm it would be appropriate to use a procedure
	- given pseudocode will use the following structure for procedure definitions:
	`PROCEDURE <identifier>`
	`  <statement(s)>`
	`ENDPROCEDURE`
	- a procedure may have none, one or more parameters
	- a parameter can be passed by reference or by value
	• show understanding of passing parameters by reference
	`PROCEDURE <identifier> (BYREF <identifier>: <datatype>)`
	`<statement(s)>`
	`ENDPROCEDURE`
	• show understanding of passing parameters by value
	`PROCEDURE <identifier> (BYVALUE <identifier>: <datatype>)`
	`<statement(s)>`
	`ENDPROCEDURE`
	- a call is made to the procedure using CALL <identifier> ()
	• use a function
	• explain where in the construction of an algorithm it is appropriate to use a function
		Give out a printed copy of a program that consists of a main routine (with a loop), a procedure and a function with a single parameter. Ask learners to label the following in the program: statement, variable, constant, condition, subroutine, procedure, function, parameter, and loop. **(I)**
		
		Explain 'call by value' and 'call by reference'. Include the underlying mechanisms (creation of local variable and value copied to it; two labels to the same item of data), effects (no change to original variable value in call by value whatever changes are made to local variable copy; any change to local variable in call by reference changes original variable value). Illustrate these ideas by running through some examples. Discuss how to handle returned values from functions. (Function result must be stored, output, or used in an expression.) **(W)**
		
		Resources:
		Built-in functions:
		http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Built-in_functions
		
		String routines:
		www.pp4s.co.uk/main/tu-stringman-routines-intro.html
		
		Basic string operations:
		www.dreamincode.net/forums/topic/82690-vb6-understanding-basic-string-operations/
		
		Procedures and functions:
		http://www.bbc.co.uk/education/guides/z9hykqt/revision
		https://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Functions_and_Procedures
		http://www.delphibasics.co.uk/Article.asp?Name=Routines

Syllabus ref	Learning objectives	Suggested teaching activities

	• use the terminology associated with procedures and functions: procedure/function header, procedure/function interface, parameter, argument, return value
	- given pseudocode will use the following structure for function definitions:
	`FUNCTION <identifier> RETURNS <data type> // function has no parameters`
	`  <statement(s)>`
	`ENDFUNCTION`
	
	`FUNCTION <identifier> (<identifier>: <data type>)`
	`RETURNS <data type> // function has one or more parameters`
	`  <statement(s)>`
	`ENDFUNCTION`
	- a function is used in an expression, for example
	o `x ← SQRT(n)`
	o `WHILE NOT EOF()`
	• write programs containing several components and showing good use of
		Use the above activities.

2.4 Software development

Syllabus ref	Learning objectives	Suggested teaching activities
2.4.1 Programming	• show understanding of how to write, translate, test and run a high-level language program • show understanding of the basic stages in the program development cycle • describe features found in a typical Integrated Development Environment (IDE): - for coding - for initial error detection, including dynamic syntax checks - for presentation, including prettyprint, expand and collapse code blocks - for debugging, including: single stepping, breakpoints, variables/expressions report window.	Discuss how modularising a problem can be beneficial in both writing and maintaining the code. The modules would be either procedures or functions and would have self-contained tasks. **(W)** Ask learners to research the stages in the program development cycle. They need to produce a diagram to represent their research and understanding of the cycle. **(I)** Demonstrate practically the range of debugging tools typically available. Translator diagnostics help with syntax and run-time error messages. Show examples. For logic errors, need to use interpreter which has the tools mentioned. **(W)** Produce a code example that has a logic error. A suitable example might be code that finds the average of a set of 100 numbers where the error is in the final arithmetic division that computes the average. A break point could be set just prior to the calculation (so that the loop does not have to be stepped through), the variables can be checked before and after the calculation which can be stepped through. In pairs, ask learners to correct the logic error(s) is the program. **(P)**

Syllabus ref	Learning objectives	Suggested teaching activities
2.4.2 Program testing	• show understanding of ways of exposing faults in programs and ways of avoiding faults • locate and identify the different types of errors: - syntax errors - logic errors - run-time errors. • correct identified errors	**Resources:** Debugging: http://msdn.microsoft.com/en-us/library/aa290042%28v=vs.71%29.aspx www.pp4s.co.uk/main/tu-debugging-intro.html www.delphibasics.co.uk/Article.asp?Name=Exceptions http://en.wikipedia.org/wiki/Error_handling Program development cycle: http://blog.teachbook.com.au/index.php/computer-science/software-development/program-development-lifecycle/

Syllabus ref	Learning objectives	Suggested teaching activities
2.4.3 Testing strategies	• choose suitable data for black-box testing • choose suitable data for white-box testing • understand the need for stub testing	Introduce the idea of black box testing: Black-box test design treats the system as a 'black-box', so it does not explicitly use knowledge of the internal code and structure. Black-box test design is usually described as focusing on testing functional requirements, external specifications or interface specifications of the program or module. **(W)** Introduce white box testing – testing all routes through a program. **(W)** Give the learners a number of small programs, with test plans that they should classify as black box or white box testing. **(I)** Introduce the concepts of stub testing when discussing structured programming and modules. **(W)** For black box testing, learners should be shown how to select inputs which are normal, borderline, and invalid. As an example for black box testing, use the following (e.g. Problem: Read two numbers, 'a' and 'b'. Put the larger of the numbers into the box 'c'). Conditions to be tested: • both numbers positive - 'a' larger - 'b' larger • one number positive - 'a' positive - 'b' positive • both numbers negative - 'a' larger (less negative) - 'b' larger • one number zero - 'a' = 0 - 'b' = 0 • both numbers equal - both positive - both negative - both zero • include other conditions.

Syllabus ref	Learning objectives	Suggested teaching activities
		Resources: Introduction to software testing – black box and white box: http://en.wikipedia.org/wiki/Software_testing www.pp4s.co.uk/main/tu-testing-intro.html http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Systems_Development_Life_Cycle/Testing

3.1 Data representation

Syllabus ref	Learning objectives	Suggested teaching activities
3.1.1 User-defined data types	• show understanding of why user-defined types are necessary • define and use non-composite types: enumerated, pointer • define and use composite data types: set, record and class/object • choose and design an appropriate user-defined data type for a given problem	This section will be covered in many other different areas, mostly object-oriented programming and file processing. Discuss non-composite and composite data types, showing examples of these. **(W)** **Resources:** http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/User-defined_data_types Enumerated types in Pascal: www.pp4s.co.uk/main/tu-enumerated-types.html Notes on pointer data type: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Pointers Pointers in Pascal: www.pp4s.co.uk/main/tu-gaming-prelim-pointers.html Notes on sets: http://en.wikipedia.org/wiki/Set_(abstract_data_type) Set data type in Pascal: www.pp4s.co.uk/main/tu-sets-intro.html Record data type in Pascal: www.pp4s.co.uk/main/tu-records-intro.html

Syllabus ref	Learning objectives	Suggested teaching activities
		Class/object notes: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Object-oriented_programming_(OOP) Classes and objects in Pascal: www.pp4s.co.uk/main/tu-oop-classes-prog.html

Syllabus ref	Learning objectives	Suggested teaching activities
3.1.2 File organisation and access	• show understanding of methods of file organisation: serial, sequential (using a key field) and random (using a record key) • show understanding of methods of file access: - sequential access for serial and sequential files - direct access for sequential and random files. • select an appropriate method of file organisation and file access for a given problem	Introduce the idea of different access methods for stored data. Relate the everyday examples such as tape recorders, CD players and playlists. Cover serial, sequential and random files and the relevant access methods: sequential and direct. **(W)** Provide learners with a set of problems about different data that needs to be stored. There should also be a description about how the data needs to be accessed. Learners should choose the most suitable method of file access for each problem. **(I)** **Resources:** Notes on sequential files: http://www.csis.ul.ie/cobol/course/SequentialFiles2.htm Notes on random access files: www.webopedia.com/TERM/R/random_access.html

Syllabus ref	Learning objectives	Suggested teaching activities
3.1.3 Real numbers and normalised floating-point representation	• describe the format of binary floating-point real numbers • convert binary floating-point real numbers into denary and vice versa • normalise floating-point numbers • show understanding of the reasons for normalisation • show understanding of the effects of changing the allocation of bits to mantissa and exponent in a floating-point representation • show understanding of how underflow and overflow can occur • show understanding of the consequences of a binary representation only being an approximation to the real number it represents • show understanding that binary representations can give rise to rounding errors	Explain the structure of a floating-point number, including definitions of the mantissa (non-zero fractional part) and exponent (integer power). **(W)** Provide examples showing the range of values that can be stored and how a normalised number allows for the greatest precision for a given size of mantissa. Explain how the increase in range leads to a decrease in precision and introduce the ideas of underflow (exponent too small) or overflow (exponent too large) as the result of a calculation. **(W)** Use method of: change to a binary number normalise the binary value adjust the exponent to accept the normalisation to create floating point representations. Set worksheet exercises to practise the conversion of a decimal number to binary floating point and binary floating-point numbers to decimal. Include positive and negative numbers, large numbers and fractional values. Give model answers to ensure correct technique. **(I)** Explain why not all numbers can be represented exactly. **(W)** Introduce the issue of errors with carefully chosen examples. **(W)** **Resources:** Theory notes for floating point numbers (sections 6–12): www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_4/floating_point/miniweb/index.htm Notes and exercises on floating point numbers: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Real_Numbers/Floating_point_numbers Notes and exercises on normalisation: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Real_Numbers/Normalisation

Syllabus ref	Learning objectives	Suggested teaching activities
		Notes on errors, underflow and overflow: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Real_Numbers/Errors

3.2 Communication and internet technologies

Syllabus ref	Learning objectives	Suggested teaching activities
3.2.1 Protocols	• show understanding of why a protocol is essential for communication between computers • show understanding of how protocol implementation can be viewed as a stack, where each layer has its own functionality	Using non-computing examples (examples from school/college would be excellent) demonstrate a need for rules for governing behaviour/communication. **(W)** Explain protocols as the rules that govern the transmission and reception of data. Briefly explain the need for both machines involved in the data transmission/reception to be configured to use the same protocols. **(W)**
	• show understanding of the function of each layer of the TCP/IP protocol • show understanding of the application of the TCP/IP protocol when a message is sent from one host to another on the internet • show understanding of how the BitTorrent protocol provides peer-to-peer file sharing • show an awareness of other protocols (HTTP, FTP, POP3, SMTP) and their purposes	Show that establishing the communication link initially is an important part of any successful communication. (Relate this to any of the non-computing examples used previously). Provide learners with partially complete TCP/IP diagrams and ask them, in pairs, to complete it. **(P)** **Resources:** Notes and exercises on protocols: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Protocols Notes and exercises on TCP/IP protocol: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/TCP/IP_protocol_stack Video about TCP/IP protocol (7:08 min): www.youtube.com/watch?v=IkKQ4lGHgqw&list=PL997A0CD223D94B27

Syllabus ref	Learning objectives	Suggested teaching activities
		Video about protocols (ignore ports) (10:54 min): www.youtube.com/watch?v=L1rtLnllTaA&list=PL997A0CD223D94B27
3.2.2 Circuit switching, packet switching and routers	• show understanding of circuit switching and where it is applicable • show understanding of packet switching • show understanding of the function of a router • explain how packet switching is used to pass messages across a network, including the internet	Describe circuit switching – a route is reserved from source to destination and the entire message is sent in order and therefore does not need to be reordered at the destination. Describe packet switching – explain the process of segmenting the message/data to be transmitted into several smaller packets. Each packet is labelled with its destination and the number of the packet. Each is despatched and many may go via different routes (routers). The original message is reassembled in the correct order at the destination. Learners could use the 'ping' and 'tracert' commands on a networked computer **(I)** **Resources:** Packet switching notes and exercises: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Packet_switching Router notes: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Internet,_Intranet_and_World_Wide_Web#Routers Internet: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Internet,_Intranet_and_World_Wide_Web#The_Internet

Syllabus ref	Learning objectives	Suggested teaching activities
		Routers: http://computer.howstuffworks.com/ethernet13.htm http://computer.howstuffworks.com/router1.htm http://computer.howstuffworks.com/router2.htm What is a packet? (two pages): http://computer.howstuffworks.com/question525.htm Packet switching: http://computer.howstuffworks.com/router3.htm

Syllabus ref	Learning objectives	Suggested teaching activities
3.2.3 Local Area Networks (LAN)	• show understanding of a bus topology network and the implications of how packets are transmitted between two hosts • show understanding of a star topology network and the implications of how packets are transmitted between two hosts • show understanding of a wireless network • explain how hardware is used to support a LAN: switch, router, servers, Network Interface Cards (NICs), wireless access points • show understanding of Ethernet and how CSMA/CD works	Use a prepared graphical interpretation of LAN systems (hopefully including the system the learners are using). Describe both the hardware and software required to enable the smooth operation. This may be better done by describing several case studies (including the system that the learners are using). **(W)** Explain how Ethernet and CSMA/CD work. **(W)** For each type of network, use large network diagrams (preferably of systems that the learners are familiar with), to help describe the three main network topologies: bus star wireless For each type describe its relative strengths and weaknesses. For example in a bus network: *Strengths* relative cost easy to install and monitor (single line) *Weaknesses* Lots of traffic down a single spine. Limitations of distance (300 m) without the need for signal boosting. If problems with the line whole system/spine segment is down. Traffic collision and the potential for monitoring network traffic from another workstation etc.

Syllabus ref	Learning objectives	Suggested teaching activities
		Resources: LANs: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Communication_and_Networking#Local_Area_Networks Ethernet: http://en.wikipedia.org/wiki/Ethernet http://computer.howstuffworks.com/ethernet6.htm CSMA/CD (pages 7–9): http://computer.howstuffworks.com/ethernet7.htm Notes on switches (pages 1–5): http://computer.howstuffworks.com/lan-switch.htm Network topologies: http://whatis.techtarget.com/definition/network-topology http://bryntegict.co.uk/resources/computing/theteacher/theory/cg3_1_2.htm

3.3 Hardware

Syllabus ref	Learning objectives	Suggested teaching activities
3.3.1 Logic gates and circuit design	• produce truth tables for common logic circuits including half adders and full adders • derive a truth table for a given logic circuit	Give your learners worksheets with different logic circuits and in pairs let them produce truth tables for these. **(P)** Learners can check their answers using a logic circuit simulator. **Resources:** Logic circuit simulator: www.kolls.net/gatesim/gatesim%20demo.swf http://www.logiccircuit.org/ Notes on half and full adders: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Uses_of_gates#Adders Detailed notes on half adders: www.circuitstoday.com/half-adder

Syllabus ref	Learning objectives	Suggested teaching activities
3.3.2 Boolean algebra	• show understanding of Boolean algebra • show understanding of De Morgan's Laws • perform Boolean algebra using De Morgan's Laws • simplify a logic circuit/expression using Boolean algebra	Explain the concepts of Boolean algebra and De Morgan's Laws. **(W)** Give your learners worksheets with logic circuits that they can express as Boolean expressions. They can then apply Boolean algebra to simplify these.**(I)** Learners should check their answers by drawing a truth table for the simplified circuit. **Resources:** Notes on Boolean algebra: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Boolean_algebra Notes on simplifying Boolean expression: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Simplifying_boolean_equations Exercises for simplifying Boolean expressions: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Boolean_identities Notes and exercises on De Morgan's Laws: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/De_Morgan%27s_Laws Brief notes on circuit minimisation: https://en.wikipedia.org/wiki/Circuit_minimization

Syllabus ref	Learning objectives	Suggested teaching activities
3.3.3 Karnaugh maps	• show understanding of Karnaugh maps • show understanding of the benefits of using a Karnaugh map • solve logic problems using Karnaugh maps	Explain the concepts of Karnaugh maps. **(W)** Give your learners worksheets with logic circuits that they can simplify using Karnaugh maps. **(I)** Learners should check their answers by drawing a truth table for the simplified circuit. **Resources:** Explanations of Karnaugh maps with simple worked examples: www.wisc-online.com/Objects/ViewObject.aspx?ID=DIG5103 www.ee.surrey.ac.uk/Projects/Labview/minimisation/karnaugh.html www.facstaff.bucknell.edu/mastascu/elessonsHTML/Logic/Logic3.html Rules of simplification: www.ee.surrey.ac.uk/Projects/Labview/minimisation/karrules.html Exercises (from Q3) on Karnaugh maps: www.allaboutcircuits.com/worksheets/k_map.html
3.3.4 Flip-flops	• show understanding of how to construct a flip-flop (SR and JK) • describe the role of flip-flops as data storage elements	Introduce the circuits for flip-flops and work through their operation as a class discussion. **(W)** Learners can research how flip-flops are used. In groups they must give a presentation to the class about their understanding of them. **(G)** **Resources:**

Syllabus ref	Learning objectives	Suggested teaching activities
		Diagram of a JK flip-flop circuit: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Fundamental_Hardware_Elements_of_Computers/Uses_of_gates#Flip_Flop Notes including SR and JK flip-flops: http://en.wikipedia.org/wiki/Flip-flop_(electronics) www.circuitstoday.com/flip-flops www.dummies.com/how-to/content/digital-electronics-types-of-flipflop-circuits.html www.indiabix.com/electronics-circuits/sr-flip-flop/ SR flip flop: www.electronics-tutorials.ws/sequential/seq_1.html JK flip flop: www.electronics-tutorials.ws/sequential/seq_2.html

Syllabus ref	Learning objectives	Suggested teaching activities
3.3.5 RISC processors	• show understanding of the differences between reduced instruction set computing processors (RISC) and complex instruction set computing processors (CISC) • show understanding of the importance/use of pipelining and registers in RISC processors • show understanding of interrupt handling on CISC and RISC processors	Introduce the concept of RISC processors and contrast with CISC processors. Provide your learners with worksheets of incomplete tables of processor characteristics. Learners to complete these in pairs. **(P)** **Resources:** Notes on CISC: http://en.wikipedia.org/wiki/Complex_instruction_set_computer Notes on RISC: http://en.wikipedia.org/wiki/RISC Detailed notes on CPU and multi-processors (including pipelining): http://en.wikipedia.org/wiki/CPU#Microprocessors RISC versus CISC explanations: http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/risc/risccisc/ www.eastaughs.fsnet.co.uk/cpu/further-ciscrisc.htm www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_3/parallel_processors/miniweb/pg7.htm Simple quiz: www.eastaughs.fsnet.co.uk/cpu/further-quiz.htm How microprocessors work: http://computer.howstuffworks.com/microprocessor.htm

Syllabus ref	Learning objectives	Suggested teaching activities
3.3.6 Parallel processing	• show awareness of the four basic computer architectures: SISD, SIMD, MISD, MIMD • show awareness of the characteristics of massively parallel computers	Define parallel processing (the simultaneous use of several processors to perform a single job). Compare this to the Von Neumann computer. Provide pre-determined scenarios of the use of parallel processing (e.g. weather forecasting, processing live images from a satellite, artificial intelligence). **(W)** **Resources:** Link to notes on parallel processing (sections 1–6): www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_3/parallel_processors/miniweb/index.htm

3.4 System software

Syllabus ref	Learning objectives	Suggested teaching activities
3.4.1 Purposes of an operating system (OS)	• show understanding of how an OS can maximise the use of resources • describe the ways in which the user interface hides the complexities of the hardware from the user	Define operating system – a set of software designed to run in the background on a computer system, giving an environment in which application software can be executed. Include the importance of HCI and the control of hardware. **(W)** *Question the learners:* • What are operating systems for (remembering the examples you have seen and worked with)? • What can all operating systems do? Reinforce the discussion about the purpose of operating systems with hand outs or notes. Using demonstration materials (including screenshots or live examples) illustrate the differences between graphical (of the various types) and command line interfaces. Ask learners to propose appropriate names for the different types, and steer them towards the correct names. **(W)** Discuss the types of user interfaces which make them appropriate for use by different types of users and in different situations. Lead the discussion with questions such as: • Why do many people dislike command line interfaces? • Who would use command line interfaces – and why? • What skills do users need to operate a graphical interface like Windows? Reinforce the class discussion with notes or hand-outs describing the characteristics of different types of user interfaces.

Syllabus ref	Learning objectives	Suggested teaching activities
	• show understanding of processor management: multiprogramming, including: - the concept of multiprogramming and a process - the process states: running, ready and blocked - the need for low-level scheduling and high-level scheduling - the concept of an interrupt - how the kernel of the OS acts as the interrupt handler and how interrupt handling is used to manage low-level scheduling	Introduce the features of operating systems that support multi-users and networking **(W)**: • memory management • scheduling Define the term interrupt (a signal from some device/source seeking the attention of the processor), the different classes of interrupt and the need to assign different priorities to interrupts (so that when two interrupts occur at the same time or an interrupt occurs whilst another is being serviced, the interrupt with the highest priority is dealt with first). **(W)** Classes of interrupt should include: • hardware failure • highest priority • program • timer • I/O • lowest priority Typical sources of interrupts should be identified including the following classes: • program generated • processor time generated • hardware failure Realise that the current program is also assigned a priority. Introduce concept of interrupt service routines and outline the sequence of actions: 1. save status (registers etc.) 2. determine cause (poll status flags) 3. take relevant action 4. restore status Explain, using diagrams on the board, the use of vectors to determine the location in memory of the appropriate routine. **(W)**

Syllabus ref	Learning objectives	Suggested teaching activities
		Introduce the concepts of jobs, processes and scheduling. **(W)** Define the terms: • job • job queue • priorities (including the concepts of processor bound and peripheral bound) • process (including running, runnable and suspended states) • scheduling Introduce scheduling and discuss the following benefits: • maximise use of hardware resources • maximise throughput • allocate resources fairly to all users • provide acceptable response time for interactive users • provide acceptable turnaround time for batch users • manage system performance (e.g. temporarily increase time taken to respond if the system is overloaded) • prevent deadlock Use simple diagrams to show the benefits of scheduling. **(W)** Include the following scheduling algorithms: • shortest job first • shortest remaining time • round robin **Resources:** Process management notes: http://en.wikipedia.org/wiki/Process_management_(computing) Scheduler notes: http://en.wikipedia.org/wiki/Two-level_scheduling http://en.wikipedia.org/wiki/Interrupt

Syllabus ref	Learning objectives	Suggested teaching activities
		Notes on interrupts: http://en.wikipedia.org/wiki/Interrupt_handler Notes on interrupts (sections 3–6): www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_1/interrupts/miniweb/index.htm
	• show understanding of paging for memory management: including: - the concepts of paging and virtual memory - the need for paging - how pages can be replaced - how disk thrashing can occur	Define the following terms: • virtual memory (include the reasons for use, e.g. allows more processes to be run than could be held in main memory) • paging Using diagrams on the board (or pre-prepared as a hand-out), explain the operation of paging in virtual memory systems. **(W)** **Resources:** Notes on virtual memory: http://en.wikibooks.org/wiki/Microprocessor_Design/Virtual_Memory www.howstuffworks.com/virtual-memory.htm Detailed theory notes on memory management: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_1/memory%20management/miniweb/index.htm Page thrashing: http://www.geekinterview.com/question_details/3333 http://web.stanford.edu/class/cs140/cgi-bin/lecture.php?topic=paging

Syllabus ref	Learning objectives	Suggested teaching activities
3.4.2 Virtual machine	• show understanding of the concept of a virtual machine • give examples of the role of virtual machines • show understanding of the benefits and limitations of virtual machines	Introduce the concept of virtual machines and let learners investigate examples of virtual machines (such as Java virtual machine). **(I)** In groups, learners should provide a large poster explaining what a virtual machine is and what the benefits and limitations of them are. **(G)** **Resources:** http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Operating_Systems/Provision_of_a_virtual_machine http://en.wikipedia.org/wiki/Virtual_machine

Syllabus ref	Learning objectives	Suggested teaching activities
3.4.3 Translation software	• show understanding of how an interpreter can execute programs without producing a translated version	Initially demonstrate the use of a compiler and the use of an interpreter. **(W)** Highlight the differences between compilation and interpretation including at a minimum: • Compiler translates the whole program (source code) into object code that can be stored and re-used. • Interpreter translates and executes a program line by line. No object code is stored for further use a program has to be translated each time it is used. Discuss the advantages and disadvantages of compilation and interpretation highlighting when it would be appropriate to use a compiler or an interpreter (e.g. use an interpreter during program development as errors can be easily checked and modified). As learners have used translators they should be able to contribute to a discussion. Provide learners with a table of characteristics and statements for interpreters and compilers. They should identify if each statement or characteristics belongs to a compiler or an interpreter. **(I)** **Resources:** Notes on interpreter: http://en.wikipedia.org/wiki/Interpreter_(computing) Notes on compilers: http://en.wikipedia.org/wiki/Compiler Detailed theory notes on translators: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_2/translators_compilers/miniweb/index.htm

Syllabus ref	Learning objectives	Suggested teaching activities
	• show understanding of the various stages in the compilation of a program: lexical analysis, syntax analysis, code generation and optimisation	Introduce the stages of compilation: **(W)** • lexical analysis • syntax analysis • code generation • optimisation Describe, in general terms, what happens during each phase including tokenisation, the use of the symbol table and handling errors. Include reference to source code and object code. Use sample code from a programming language that your learners are familiar with to demonstrate the general principles. **Resources:** Detailed notes on compilation: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_2/lexical_syntax_analysis/miniweb/index.htm Lexical analysis: http://en.wikipedia.org/wiki/Lexical_analysis Syntax analysis: http://en.wikipedia.org/wiki/Syntax_analysis

Syllabus ref	Learning objectives	Suggested teaching activities
	• show understanding of how the grammar of a language can be expressed using syntax diagrams or Backus−Naur Form (BNF) notation	Demonstrate the use of syntax diagrams as a formal method to describe simple syntax of a set of rules. **(W)** Demonstrate on the board the use of Backus−Naur form (BNF) as a formal method to describe simple syntax of a programming language. **(W)** Use the following meta symbols: • ::= is defined by • | OR ♦ meta variable • e.g. • <hexdigit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F Learner-centred exercise using worksheets to reinforce / test knowledge – perhaps providing simple examples to extend. Revise the answers to the worksheet as a class discussion to reinforce the concepts studied.**(I)** **Resources:** Notes on syntax diagrams: http://en.wikipedia.org/wiki/Syntax_diagram Notes on BNF: http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form Detailed notes on syntax diagrams and BNF: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_7/bnf/miniweb/index.htm

Syllabus ref	Learning objectives	Suggested teaching activities
	• show understanding of how Reverse Polish notation (RPN) can be used to carry out the evaluation of expressions	Using some examples get your learners to suggest why infix expressions present problems for translators. Show how an HLL expression might be represented as a set of assembly language statements (to illustrate problems of brackets and order of evaluation). Do the same with the equivalent reverse Polish expression to bring out the advantages of this form of the expression. **(W)** Demonstrate how a particular tree traversal method can produce the infix form of an expression. Demonstrate clearly (have a succession of stacks rather than just one) to show how the stack contents change when a reverse Polish string of characters is processed. **(W)** Give your learners a prepared sheet of exercises with empty stacks to encourage the correct layout of answers. **(I)** **Resources:** Notes on Reverse Polish notation: http://en.wikipedia.org/wiki/Reverse_Polish_notation Notes on Reverse Polish notation: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_7/revpolish/miniweb/index.htm

3.5 Security

Syllabus ref	Learning objectives	Suggested teaching activities
3.5.1 Asymmetric keys and encryption methods	• show understanding of the terms: public key, private key, plain text, cipher text, encryption and asymmetric key cryptography • show understanding of how the keys can be used to send a private message from the public to an individual/organisation • show understanding of how the keys can be used to send a verified message to the public	Start with a review of the security topic from Section 1.6. Discuss how secure symmetric encryption is and the reason for asymmetric encryption. **(W)** Develop, through class discussion, how public and private keys are used to send encrypted messages. **(W)** Provide the learners with partially complete diagrams they can complete. **(I)** **Resources:** Encryption: www.howstuffworks.com/encryption.htm http://en.wikipedia.org/wiki/Ciphertext http://en.wikipedia.org/wiki/Public-key_cryptography Good diagram of private/public key usage (Alice and Bob): http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Communication_and_Networking#Internet_Security

Syllabus ref	Learning objectives	Suggested teaching activities
3.5.2 Digital signatures and digital certificates	• show understanding of how a digital certificate is acquired • show understanding of how a digital certificate is used to produce digital signatures	Class discussion of the issue of ensuring that information is from a trusted source. Introduce scenarios where a message is from an impersonator, or the message got maliciously changed during transit. **(W)** Introduce the use of digital certificates to verify the authenticity of the message sender and provide the receiver with the means to encode a reply. **(W)** Provide the learners with partially complete diagrams they can complete.**(I)** **Resources:** What is a digital signature?: http://computer.howstuffworks.com/digital-signature.htm Digital certificate (aka public key certificate): http://en.wikipedia.org/wiki/Public_key_certificate Contents of a typical digital certificate: http://en.wikipedia.org/wiki/Public_key_certificate#Contents_of_a_typical_digital_certificate Diagrams of how digital signatures and digital certificates are used: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Communication_and_Networking#Digital_signatures

Syllabus ref	Learning objectives	Suggested teaching activities
3.5.3 Encryption protocols	• show awareness of the purpose of Secure Socket Layer (SSL)/Transport Layer Security (TLS) • show awareness of the use of SSL/TLS in client-server communication • show awareness of situations where the use of SSL/TLS would be appropriate	Class discussion on where SSL is used and how the user of a web page can tell whether the communication link is secure. **(W)**
3.5.4 Malware	• show understanding of malware: viruses, worms, phishing, pharming • describe vulnerabilities that the various types of malware can exploit • describe methods that can be used to restrict the effect of malware	Give learners a set of term and definitions, they need to match them. **(I)** Discuss how the effect of malware can be restricted. **(W)** **Resources:** Avoiding malware: http://www.pcworld.com/article/210891/malware.html https://zeltser.com/malware-in-the-enterprise/

3.6 Monitoring and control systems

Syllabus ref	Learning objectives	Suggested teaching activities
3.6.1 Overview of monitoring and control systems	• show understanding of the difference between a monitoring system and a control system • show understanding of sensors and actuators and their usage • show understanding of the additional hardware required to build these systems • show understanding of the software requirements of these systems • show understanding of the importance of feedback in a control system	Introduce sensors in a real-time computer system and how this constitutes a **monitoring** system.**(W)** Introduce the idea of a feedback loop by describing a simple system e.g. a temperature **control** system attached to a heater and a fan. Also discuss the need for sensors and actuators to implement this system. **(W)** Extend this work to look at a variety of other real time systems that use the following types of signals: • visible • tactile • audible • other physical signals **Resources:** http://en.wikipedia.org/wiki/Real-time_computing http://en.wikipedia.org/wiki/Control_system http://en.wikipedia.org/wiki/Sensor http://en.wikipedia.org/wiki/Actuator www.bbc.co.uk/schools/gcsebitesize/ict/measurecontrol/0computercontrolrev1.shtml

Syllabus ref	Learning objectives	Suggested teaching activities
3.6.2 Bit manipulation to monitor and control devices	• show understanding of how bit manipulation can be used to monitor/control a device • carry out bit manipulation operations: test a bit and set a bit (using bit masking) using the instructions from Section 1.4.3 and those listed below • show understanding of how to make use of appropriate bit manipulation in monitoring systems and control systems	Demonstrate the operation of AND, OR, XOR on bit patterns. **(W)** Provide learners with different scenarios, such as: A vending machine where bit positions determine the type of beverage dispensed: • Bit 0: tea • Bit 1: coffee • Bit 2: chocolate • Bit 3: milk • Bit4: sugar Learners set a byte with the relevant bits to 1 to dispense **(I)**: • coffee, no milk and with sugar • tea with milk, no sugar **Resources:** Notes and exercises for bit manipulation: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Logical_bitwise_operators

4.1 Computational thinking and problem-solving

Syllabus ref	Learning objectives	Suggested teaching activities
4.1.1 Abstraction	• show understanding of how to model a complex system by only including essential details, using: - functions and procedures with suitable parameters (as in procedural programming, see Section 2.3) - ADTs (see Section 4.1.3) - classes (as used in object-oriented programming, see Section 4.3.1) - facts, rules (as in declarative programming, see Section 4.3.1)	Learning resource 'Abstraction notes' give very good introduction, especially under headings: • structured programming • data abstraction • abstraction in object-oriented programming (OOP) This topic should be interwoven into the different topics of ADTs, OOP, declarative programming. **Resources:** Definition of computational thinking and links to further reading: http://en.wikipedia.org/wiki/Computational_thinking What is computational thinking? And links to further reading: www.google.com/edu/computational-thinking/index.html http://www.bbc.co.uk/education/topics/z7tp34j Abstraction notes: http://en.wikipedia.org/wiki/Data_abstraction The four parts of computational thinking: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Introduction_to_principles_of_computation Abstraction and intro to OOP using Pascal: www.delphibasics.co.uk/Article.asp?Name=Abstract

Syllabus ref	Learning objectives	Suggested teaching activities
4.1.2 Algorithms	• write a binary search algorithm to solve a particular problem • show understanding of the conditions necessary for the use of a binary search • show understanding of how the performance of a binary search varies according to the number of data items	Demonstrate the use of linear and binary searches with several sets of data. Choose the data sets very carefully to show the advantages and disadvantages of each type of search by using both algorithms on the same set of data. **(W)** Provide learners with a binary search algorithm and ask them, in pairs, to comment what each part of it is doing. **(P)**
	• write an algorithm to implement an insertion sort • write an algorithm to implement a bubble sort	Demonstrate the following sort routines **(W)**: • bubble sort • insertion sort Provide learners with an example of each sorting algorithm and ask them, in pairs, to comment on what each part of it is doing. **(P)**
	• show understanding that performance of a sort routine may depend on the initial order of the data and the number of data items • write algorithms to find an item in each of the following: linked list, binary tree, hash table • write algorithms to insert an item into each of the following: stack, queue, linked list, binary tree, hash table • write algorithms to delete an item from each of the following: stack, queue, linked list	The standard algorithms for each type of ADT should be covered when teaching the relevant ADT. **Resources:** Links to theory notes for searching and sorting: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_5/data_structures/miniweb_search/index.htm Animation of many different sort algorithms: www.sorting-algorithms.com/ Notes and exercises for bubble sort and linear search: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Searching_and_sorting

Syllabus ref	Learning objectives	Suggested teaching activities
		Notes and exercises for insertion sort: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Insertion_sort Notes and exercises for binary search: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Binary_search
	• show understanding that different algorithms which perform the same task can be compared by using criteria such as time taken to complete the task and memory used	Although BigO notation is not part of this syllabus, the resource listed here gives an insight into the efficiency of different algorithms. Present learners with different algorithms to do the same task. They could then program these and run them to compare speeds. **(I)** **Resources:** http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Problem_Solving/BigO_notation

Syllabus ref	Learning objectives	Suggested teaching activities

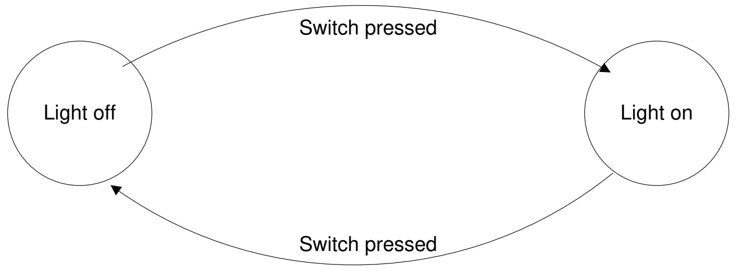
4.1.3 Abstract Data Types (ADT)	• show understanding that an ADT is a collection of data and a set of operations on those data
	
	• show understanding that data structures not available as built-in types in a particular programming language need to be constructed from those data structures which are built-in within the language
	
	`TYPE <identifier1>`
	  `DECLARE <identifier2> : <data type>`
	  `DECLARE <identifier3> : <data type>`
	  …
	`ENDTYPE`
	• show how it is possible for ADTs to be implemented from another ADT
	
	• describe the following ADTs and demonstrate how they can be implemented from appropriate built-in types or other ADTs: stack, queue, linked list, dictionary, binary tree
		Introduce each ADT separately using the notes and exercises listed in the learning resources.
		
		To reinforce the concepts, learners should write programs using the data structures. **(I)**
		
		ADTs are usually implemented from the built-in data type ARRAY.
		
		In OOP, classes could be declared with subclasses to implement different ADTs. For example, stacks and queues are special types of linked list.
		
		Resources:
		
		Definition of ADT:
		http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Abstract_data_types_and_data_structures
		
		Notes on ADTs:
		www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_5/data_structures/miniweb/index.htm

Syllabus ref	Learning objectives	Suggested teaching activities
4.1.4 Recursion	• show understanding of the essential features of recursion • show understanding of how recursion is expressed in a programming language • trace recursive algorithms • write recursive algorithms • show understanding of when the use of recursion is beneficial • show awareness of what a compiler has to do to implement recursion in a programming language	Discuss the nature of recursion: a subroutine that calls itself; to succeed it needs a stopping condition. Show some definitions that are suitable for solution by recursive algorithms. Discuss where in the algorithms the recursion occurs and also highlight and discuss the stopping conditions. **(W)** Give your learners a couple of recursive subroutines and ask them to highlight the recursive calls and also the stopping conditions. Include some examples that have recursive calls or stopping conditions omitted. **(I)** Being able to trace successfully a recursive subroutine is very helpful in grasping recursion. Use a diagrammatic method of tracing which clearly shows: the 'recursive descent' until the stopping condition is encountered; the return of values as the recursion unwinds. Factorial and Fibonacci are suitable examples to demonstrate. **(W)** Give learners some questions (include some non-mathematical examples, e.g. printing a list of items). Check their answers. **(I)** Compare iterative and recursive algorithms for a couple of problems. Discuss size of solution, elegance of solution, run-time memory requirements and speed of execution with regard to the two alternative versions of a solution. **(W)** **Resources:** Definition of recursion + lots of examples: http://en.wikipedia.org/wiki/Recursion_%28computer_science%29 Notes and exercises: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Recursive_Techniques Iterative versus recursive notes: www.codeproject.com/Articles/21194/Iterative-vs-Recursive-Approaches

4.2 Algorithm design methods

Syllabus ref	Learning objectives	Suggested teaching activities
4.2.1 Decision tables	• describe the purpose of a decision table • construct a decision table for a given problem with a maximum of three conditions • simplify a decision table by removing redundancies	Introduce the concept of decision tables. **(W)** Present learners with sets of logical conditions and actions for them to produce decision tables. Give more complicated logic scenarios to give rise to table simplification. Solutions could be programmed and checked for correctness using comprehensive test data. **(I)** **Resources:** Decision tables: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Decision_tables Example of decision table and how to simplify: http://webfuse.cqu.edu.au/Courses/2009/T1/COIT11226/Resources/Additional_Resources/Decision%20Table%20Example.htm

Syllabus ref	Learning objectives	Suggested teaching activities
4.2.2 Jackson Structured Programming (JSP)	• construct a JSP structure diagram showing repetition • construct a JSP structure diagram showing selection • write equivalent pseudocode from such structure charts • construct a JSP structure diagram to describe a data structure • construct a JSP data structure diagram: - using sequence - using selection - using iteration • construct a JSP diagram for a program design	Provide simple problems for learners to draw JSP structure diagrams. **(W)** Give a prepared JSP structure diagram to learners to write equivalent program/pseudocode. **(I)** **Resources:** Notes including a worked example: http://en.wikipedia.org/wiki/Jackson_structured_programming

Syllabus ref	Learning objectives	Suggested teaching activities
4.2.3 State-transition diagrams	• use state-transition diagrams to document an algorithm • use state-transition diagrams to show the behaviour of an object	A finite state machine (FSM) is a machine that consists of a set of possible states. Inputs change the state of the FSM. To show this diagrammatically, a state-transition diagram (STD) is used. Sometimes these are just called state diagrams. For example, a desk lamp has two states: light on light off The input is to press the switch. This would be shown diagrammatically:  Give learners simple FSMs to draw diagrams. **(I)** **Resources:** Detailed introduction to state-transition diagrams: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Finite_state_machines www.nikhef.nl/~p63/www/STD.html Turnstile example: http://en.wikipedia.org/wiki/Finite-state_machine More examples (ignore directed graph topic): http://en.wikipedia.org/wiki/State_diagram http://en.wikipedia.org/wiki/State_transition_table

4.3 Further programming

Syllabus ref	Learning objectives	Suggested teaching activities
4.3.1 Programming paradigms	• show understanding of what is meant by a programming paradigm • show understanding of the characteristics of a number of programming paradigms (low-level, imperative (procedural), object-oriented, declarative) - low-level programming o demonstrate an ability to write low-level code that uses various address modes: immediate, direct, indirect, indexed and relative (see Section 1.4.3 and Section 3.6.2) - imperative programming o see details in Section 2.3 (procedural programming) - object-oriented programming (OOP) o demonstrate an ability to solve a problem by designing appropriate classes o demonstrate an ability to write code that demonstrates the use of classes, inheritance, polymorphism and containment (aggregation)	Provide term and definition cards for definitions of the following types of programming languages and the characteristics of each: • declarative • procedural • object oriented • low level Learners need to match the correct term and definition. **(I)** Explain the concepts of object-oriented languages including at a minimum **(W)**: • encapsulation (keeping together data structures and methods) • classes • derived classes • inheritance (derived classes carry the data structures and methods of the superclass) • polymorphism • containment/aggregation Use everyday examples to introduce these ideas, e.g. class definition of clock, derived classes – analogue clock and digital clock. Show how classes and inheritance can be represented on an **inheritance diagram** by using a number of examples. Show examples of **object diagrams**. Highlight the differences between the two types of diagram and use this to reinforce the difference between a class and an object **(W)**. Learners should have practical experience of programming using OOP **(I)**.

Syllabus ref	Learning objectives	Suggested teaching activities
	- declarative programming ○ demonstrate an ability to solve a problem by writing appropriate facts and rules based on supplied information ○ demonstrate an ability to write code that can satisfy a goal using facts and rules	Explain the concepts of declarative languages including at a minimum **(W)**: • rules • facts • backtracking • instantiation (binding of a variable to a value during resolution, lasting only long enough to satisfy one complete goal) • satisfying goals **Resources:** Programming paradigms: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Programming_paradigms Notes and exercises on OOP including inheritance diagrams: http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Programming_Concepts/Object-oriented_programming_(OOP) Link to notes on different programing paradigms: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_6/types_language/miniweb/index.htm OOP programming using VB: www.studyvb.com/Object-Oriented-Programming.html OOP programming using Pascal: www.pp4s.co.uk/main/tu-oop-intro.html www.delphibasics.co.uk/Article.asp?Name=OO

Syllabus ref	Learning objectives	Suggested teaching activities
		OOP programming with Python: www.codecademy.com/courses/python-intermediate-en-WL8e4?curriculum_id=4f89dab3d788890003000096 Object diagram notes: http://en.wikipedia.org/wiki/Object_diagram Links to theory notes: www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_6/declarative/miniweb/index.htm Prolog (free downloads): www.learnprolognow.org/lpnpage.php?pageid=implementations Tutorial guide to prolog: www.learnprolognow.org/lpnpage.php?pageid=online

Syllabus ref	Learning objectives	Suggested teaching activities

4.3.2 File processing (see also Section 2.2.3)	• write code to define a record structure
	• write code to perform file-processing operations: open or close a file; read or write a record to a file
	• use pseudocode for random file handling:
	`OPENFILE <filename> FOR RANDOM`
	`SEEK <filename>, <address> // move a pointer to the disk address for the record`
	`GETRECORD <filename>, <identifier>`
	`PUTRECORD <filename>, <identifier>`
	• write code to perform file-processing operations on serial, sequential and random files
		Introduce learners to files of records, initially just writing records out to file, then reading them back into an array (serial and sequential files). **(W)**
		
		Then introduce direct access to a file of records (random files). **(W)**
		
		Learners should create a small file handling program. **(I)**
		
		Note: contents of files of records cannot easily be checked in a text editor as non-string data types will not be represented correctly.
		
		Resources:
		Notes on records (user-defined types):
		http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/User-defined_data_types
		
		File handling in Pascal:
		www.pp4s.co.uk/main/tu-records-files.html
		www.pp4s.co.uk/main/tu-io-infile.html
		www.pp4s.co.uk/main/tu-io-outfile.html
		
		Record type in Visual Basic (VB):
		http://visualbasic.freetutes.com/learn-vb6/lesson6.1.html
		
		File handling in VB:
		www.dreamincode.net/forums/topic/56171-file-handling-in-visual-basic-6-part-2-binary-file-handling/

Syllabus ref	Learning objectives	Suggested teaching activities
4.3.3 Exception handling	• show understanding of an exception and the importance of exception handling • show understanding of when it is appropriate to use exception handling • write code to use exception handling in practical programming	Discuss with learners the importance of software that does not crash. **(W)** Get learners to write programs using simple exception handling using the construct TRY … EXCEPT. **(I)** **Resources:** Exception handling using Python: www.pythonforbeginners.com/error-handling/how-to-handle-errors-and-exceptions-in-python/ Exception handling in VB: www.dreamincode.net/forums/topic/82982-error-handling-in-vb/ Exception handling with Pascal: www.pp4s.co.uk/main/tu-debugging-errorhandling.html
4.3.4 Use of development tools/programming environments	• describe features in editors that benefit programming • know when to use compilers and interpreters • describe facilities available in debuggers and how and when they should be deployed	Get learners to discuss the features in a chosen development environment. **(W)** Get learners to find out which debugging features listed in http://en.wikipedia.org/wiki/Debugger are available in their development environment. They should have practical experience of using these with suitable program code (pre-prepared) **(I)**: • stepping • variable watch • breakpoints Discuss which translator is more appropriate if both compiler and interpreter exist for a programming language. **(W)**

4.4 Software development

Syllabus ref	Learning objectives	Suggested teaching activities
4.4.1 Software development resources	• show understanding of the possible role of program generators and program libraries in the development process	Learners should research program generators and program libraries. They should comment on how they can be used in development. **(I)** **Resources:** Generators: http://www.computerhope.com/jargon/p/proggene.htm https://en.wikipedia.org/wiki/Generator_(computer_programming) Libraries: https://en.wikipedia.org/wiki/Library_(computing) https://www.techopedia.com/definition/3828/software-library http://searchsqlserver.techtarget.com/definition/library

Syllabus ref	Learning objectives	Suggested teaching activities
4.4.2 Testing	• show awareness of why errors occur • show understanding of how testing can expose possible errors • appreciate the significance of testing throughout software development • show understanding of the methods of testing available: dry run, walkthrough, white-box, black-box, integration, alpha, beta, acceptance • show understanding of the need for a test strategy and test plan and their likely contents • choose appropriate test data (normal, abnormal and extreme/boundary) for a test plan	Introduce the concepts of integration testing, alpha testing and beta testing. These are user tests. Explain that the programmer tests focus on error-free processing. User tests focus on usability, functionality, and performance. User testing with test data is called alpha testing. This is then followed by beta testing during which users use the system with their own data. **(W)** Introduce acceptance testing. This is the final test by the customer to check that the developed system is what they asked for. **(W)** Get learners to complete some dry runs for given algorithms. Using trace tables can work best. **(I)** **Resources:** http://www.bbc.co.uk/education/guides/z8n3d2p/revision/8 http://blog.teachbook.com.au/index.php/computer-science/software-development/trace-tables/

Syllabus ref	Learning objectives	Suggested teaching activities
4.4.3 Project management	• show understanding that large developments will involve teams • show understanding of the need for project management • show understanding of project planning techniques including the use of GANTT and Program Evaluation Review Technique (PERT) charts • describe the information that GANTT and PERT charts provide • construct and edit GANTT and PERT charts	Explain to learners what a GANTT chart and a PERT chart is and how they are used. **(W)** Provide learners with a list of tasks for completing and activity, such as building a house. The activities will need time scales and dependents. Ask learners to create a GANTT chart for the project. **(I)** **Resources:** GANTT chart: http://en.wikipedia.org/wiki/Gantt_chart PERT chart: http://en.wikipedia.org/wiki/Pert