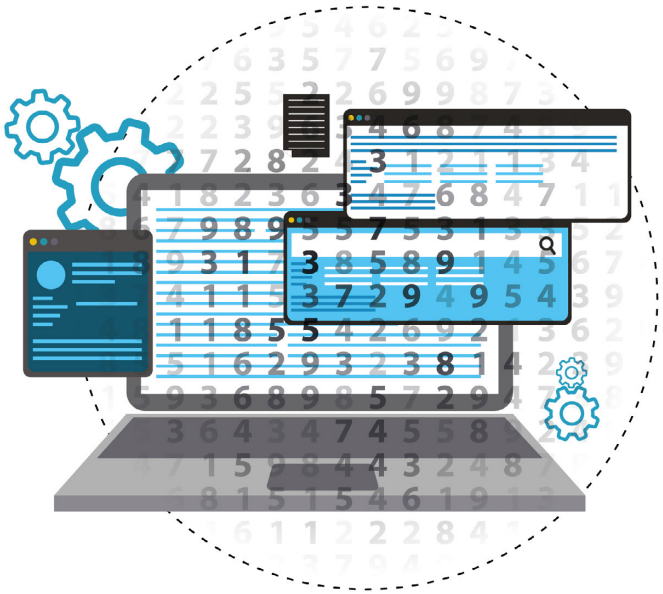
Cambridge Assessment International Education

Scheme of Work

Cambridge International AS & A Level Computer Science 9618

For examination from 2021



Version 2

Cambridge Pathway

Contents

Introduction

This scheme of work has been designed to support you in your teaching and lesson planning. Making full use of this scheme of work will help you to improve both your teaching and your learners' potential. It is important to have a scheme of work in place in order for you to guarantee that the syllabus is covered fully. You can choose what approach to take and you know the nature of your institution and the levels of ability of your learners. What follows is just one possible approach you could take and you should always check the syllabus for the content of your course.

Suggestions for independent study **(I)** and formative assessment **(F)** are also included. Opportunities for differentiation are indicated as **Extension activities**; there is the potential for differentiation by resource, grouping, expected level of outcome, and degree of support by teacher, throughout the scheme of work. Timings for activities and feedback are left to the judgement of the teacher, according to the level of the learners and size of the class. Length of time allocated to a task is another possible area for differentiation.

Key concepts

This scheme of work is underpinned by the assumption that Computer Science is a practical subject and learners should be engaged in practical activities throughout the course. The key concepts are highlighted as a separate item in the new syllabus. Reference to the key concepts is made throughout the scheme of work using the key shown below.

Key Concept 1 (KC1) – Computational thinking

Computational thinking is a set of fundamental skills that help produce a solution to a problem. Skills such as abstraction, decomposition and algorithmic thinking are used to study a problem and design a solution that can be implemented. This may involve using a range of technologies and programming languages.

Key Concept 2 (KC2) – Programming paradigms

A programming paradigm is a way of thinking about or approaching problems. There are many different programming styles that can be used, which are suited to unique functions, tools and specific situations. An understanding of programming paradigms is essential to ensure that they are used appropriately, when designing and building programs.

Key Concept 3 (KC3) – Communication

Communication is a core requirements of computer systems. It includes the ability to transfer data from one device or component to another and an understanding of the rules and methods that are used in this data transfer. Communication could range from the internal transfer of data within a computer system, to the transfer of a video across the internet.

Key Concept 4 (KC4) – Computer architecture and hardware

Computer architecture is the design of the internal operation of a computer system. It includes the rules that dictate how components and data are organised, how data are communicated between components, to allow hardware to function. There is a range of architectures, with different components and rules, that are appropriate for different scenarios.

All computers comprise of a combination of hardware components, ranging from internal components, such as the Central Processing Unit (CPU) and main memory, to peripherals. To produce effective and efficient programs to run on hardware, it is important to understand how the components work independently and together to produce a system that can be used. Hardware needs software to be able to perform a task. Software allows hardware to become functional. This enables the user to communicate with the hardware to perform tasks.

Key Concept 5 (KC5) – Data representation and structures

Computers use binary and understanding how a binary number can be interpreted in many different ways is important. Programming requires an understanding of how data can be organised for efficient access and/or transfer.

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 hours for Cambridge International AS Level, and 360 hours for Cambridge International A Level. The number of hours may vary depending on local practice and your 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 (hours)	Suggested teaching order
1 Information representation	12	1
2 Communication	16	4
3 Hardware	11	2
4 Processor Fundamentals	15	3
5 System Software	8	7
6 Security, privacy and data integrity	8	6
7 Ethics and Ownership	6	5
8 Databases	18	9
9 Algorithm Design and Problem-Solving	28	Continuous through AS Level
10 Data Types and structures	22	Continuous through AS Level

Topic	Suggested teaching time (hours)	Suggested teaching order
11 Programming	24	Continuous through AS Level
12 Software Development	12	8
13 Data Representation	15	10
14 Communication and internet technologies	15	13
15 Hardware and Virtual Machines	15	11
16 System Software	15	12
17 Security	10	14
18 Artificial Intelligence (AI)	10	15
19 Computational thinking and problem solving	50	Continuous through A Level
20 Further Programming	50	Continuous through A Level

Resources

You can find the endorsed resources to support Cambridge International AS & A Level Computer Science 9618 on the Published resources tab of the syllabus page on our public website here.

Endorsed textbooks have been written to be closely aligned to the syllabus they support, and have been through a detailed quality assurance process. All textbooks endorsed by Cambridge International for this syllabus are the ideal resource to be used alongside this scheme of work as they cover each learning objective. In addition to reading the syllabus, teachers should refer to the specimen assessment materials.

School Support Hub

The School Support Hub www.cambridgeinternational.org/support 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 PDF and an editable version in Microsoft Word format; both are available on the School Support Hub at www.cambridgeinternational.org/support If you are unable to use Microsoft Word you can download Open Office free of charge from www.openoffice.org

Websites

This scheme of work includes website links providing direct access to internet resources. Cambridge Assessment International Education 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 website pages referenced in 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 9618 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.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Use Backus-Naur Form (BNF) and syntax diagrams to express the grammar of a language. Use Reverse Polish Notation (RPN) to carry out the evaluation of expressions.	Give learners program code and a BNF diagram and ask them whether the code is valid/invalid and to justify answers. Give learners a scenario and ask them to create a BNF diagram from it. **(I)** Give learners a BNF diagram and ask them to change it, to introduce more options, and/or restrict the gram further. **(F) (I)** **Extension:** Ask learners to write a program to check the syntax of a program that is input against BNF statements. Discuss the use of brackets and priorities in mathematical calculations. Ask learners how computers 'know action to perform first. Show learners how RPN can tell computers the order to perform the calculations in learners how stacks are used to represent the RPN expression and how these are evaluated. Give learners blank 'stack' diagram and a RPN expression; ask them to put the items on the stack and then perform the calculations one step at a time – getting the item(s) from the stack, performing the calculation and putting t output back. Give learners a mathematical expression and ask them to put it into RPN. Repeat but reversed, giving learners an RPN expression and turn it into a normal mathematical expression. **Extension:** Ask learners to write a program that takes an expression and produces the RPN expression for it.

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

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

Past and specimen papers
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)**

9618/3	*SAM*	*Q4*
9608/32	*June-2018*	*Q5*
9608/31	*Oct-2016*	*Q2*
9608/31	*Oct-2016*	*Q2, 3*
9608/32	*Oct-2016*	*Q2, 3*
9608/31	*Oct-2017*	*Q3*
9608/32	*Oct-2017*	*Q3*

Formative assessment (F) is ongoing 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, 'mind maps', or 'concept maps'. These kinds of activities can be found in the scheme of work.

Past papers, specimen papers and **mark schemes** are available for you to download at: www.cambridgeinternational.org/support

Using these resources with your learners allows you to check their progress and give them confidence and understanding.

Unit 1 Information Representation

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
1.1 Data Representation (binary) **(KC5)**	Convert a number from one base to another. Perform binary additional and subtraction. Explain the purpose and benefits of different number bases.	Learners should practise converting from one number base to another. Start with unsigned 8-bit binary: ask learners to count using binary for example starting with 0, then writing the binary for 1, 2, 3, etc. Challenge learners to work out the largest number they can represent with a set number of bits, e.g. 16 bits. **(F)** Ask learners to also identify the quantity of different numbers that can be represented and how this is different to the largest number. **(I)** **Extension:** Ask learners to program a binary number generator that creates binary numbers for learners to work out and then checks if they are correct. This can be extended with other number bases. **(I)** Show learners a large binary number and ask learners to copy it and what the difficulties are: e.g. easy to make mistakes, difficult to remember. Show learners BCD as an alternative for representing a binary number and ask learners why this could be beneficial. Repeat with the hexadecimal representation and ask learners to research and find examples of where hexadecimal is used in computers. **(I)** Ask learners what the limitations are of unsigned binary; i.e. it can only represent positive whole numbers. Show learners how to use ones' and two's complement to represent negative numbers. Show learners how to add binary numbers, introducing the rules of addition and why these work. Explore overflow in addition and then how subtraction can just be the addition of a negative number. **Extension:** Ask learners to explore how a computer handles overflow, e.g. interrupts, additional registers. **(I)**
1.1 Data representation (character sets) **(KC5)**	Explain the use of character sets in computer systems. Use ASCII, extended ASCII and Unicode to represent textual data.	Provide learners with an ASCII table (or other character set) and ask them to write a message to another learners. Ask learners to swap their messages and decipher them. Discuss the need for character sets in computer systems and why different sets exist. Ask learners to explore the difference between ASCII and extended ASCII, then repeat with Unicode. Discuss the benefits and drawbacks of each, e.g. file size and range of characters. Show learners the relationship between characters, e.g. how the binary values increase for consecutive values. **Extension:** Ask learners to develop their own character set and write a computer program to read the binary (or hexadecimal) value and display the characters. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
1.2 Multimedia (*Graphics*) **(KC5)**	Explain how a bitmap image is represented and stored on a computer. Explain how a vector graphic is represented and stored on a computer. Explain whether a bitmap image of vector graphic is more appropriate for a given task.	Give learners a grid with a number in each square and a table with the numbers and colours. Ask learners to use the table and grid to colour the image. Repeat with the numbers for the grid separate, in binary (e.g. 4 bits per square) and in one long string. Discuss how the grid represents a bitmap image and how the colours are represented by binary numbers. Link back to binary numbers and how many combinations you can have for a set number of bits, e.g. 4 bits can have 16 different combinations. Ask learners how this relates to the number of colours. Students should maintain a glossary of key terms for graphics. Add colour depth, bit depth, etc., as they are covered. **(F)** Provide learners with a set of instructions to create an image – e.g. coordinates to draw lines, colours to fill – to represent how a vector graphic is stored. Students should add the key vector graphic terms to their glossary. Ask learners to compare the two methods of storing images and the relative merits and drawbacks. Provide learners with a scenario and ask them to work in pairs or groups to identify whether a vector or bitmap is more appropriate and to justify their choice. **(I)** **Extension:** Students can develop an interactive quiz of key terms, for example as a program or by integrating code into a PowerPoint presentation. **(F)** **Extension:** Explore image manipulation software, how bitmap and vector graphics can be edited, and what restrictions there are in each type of image. **(I)**
1.2 Multimedia (*Sound*) **(KC5)**	Explain how an analogue sound wave is digitised. Explain the effect of changing the sample rate and resolution on a sound wave.	Give learners a graph with axis and a series of numbers. Ask learners to plot the numbers on the graph and then join these to create a digital sound wave. Show learners the original analogue wave and explore the differences between them. Students should maintain a glossary of key terms for sound theory. **(F)** Give learners a diagram of an analogue sound wave with amplitudes on the y axis. Give them an example sample rate and ask them to recreate the analogue wave in digital. Repeat this with different sample rates and ask them to explain the differences between the sound waves and how these would impact the file size and resulting sound. **Extension:** Ask learners to record a sample sound in software such as Audacity (www.audacityteam.org), and to them manipulate the sound wave. Ask them to record the sound at different sample rates, to change the wave and see what the results are. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
1.3 Compression **(KC5)**	Explain the need for compression. Explain the difference between lossy and lossless compression. Recommend lossy or lossless compression for a given scenario and justify the choice. Show how a sound/image/text can be compressed using run-length encoding.	Ask learners what they do when they need to send a file by email but it is too large – do they upload it to the cloud? Change the format? Compress it? Show learners an example of lossy and lossless compression on the same document (e.g. an image) before and after each method and explain the difference in the results. Give learners a list of documents, e.g. image for a website, program code, text document. Ask learners to decide whether each one should be compressed using lossy or lossless compression, and why. Give learners a specific type of file (e.g. text, image, sound) and ask them to find examples of how they can be compressed using lossy/lossless, and to then describe these to the rest of the learners. **(I)** Give learners an image that has been encoded using RLE, for example 3B, 4R, 2Y and ask them what they think it means. Ask them if they can replicate the image from the code. **(I)** Show learners how run-length encoding works on different types of file, e.g. images, text files, etc. **Extension:** Students can explore other methods of compression, e.g. Huffman encoding. **(I)** https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/ https://people.ok.ubc.ca/ylucet/DS/Huffman.html **Extension:** Students can write a program to compress a string (or text file) using run-length encoding **(I)** https://www.techiedelight.com/run-length-encoding-rle-data-compression-algorithm/
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/1 Specimen paper Q1		

Unit 2 Communication

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
2.1 Networks including the internet (introduction to types of network) **(KC3)**	Explain the purpose and benefits of networking devices. Describe the characteristics of a LAN and a WAN. Explain whether a given network is a LAN or a WAN. Describe the use, benefits and drawbacks of cloud computing. Describe the characteristics of a client-server and peer-to-peer network. Explain the benefits and drawbacks of a client-server and peer-to-peer network. Justify the use of a client-server or peer-to-peer network in a given scenario.	Ask learners to identify all the different tasks they perform using a network each day, e.g. a school network, home network, mobile phone, any access to the internet, etc. Discuss the answers and collate these into examples of the purpose and benefits of networking devices. **Extension:** Ask learners to identify the purpose and benefits of different devices; e.g. is there a different purpose/benefit or connecting a printer to a network than a mobile phone? **(I)** Introduce LAN and WAN concepts and the characteristics of the different sizes, and the ownership of the hardware. Show learners a list of scenarios and ask them to vote for whether each one is a LAN or a WAN. Select some learners to explain their choices (there may not be a right and wrong answer, because there may be insufficient information). **(F)** Split learners into groups and give some groups client-server, and some peer-to-peer models. Ask the groups to find out a) a definition of the model they have been given, b) the benefits of this model, c) the drawbacks of this model, d) three different examples of where this model is used. **(I)** Collate the answers from all the groups, for example from one group take the definition, then ask the second if there is anything to add/change/refine, etc. Repeat until there is a comprehensive guide to client-server and peer-to-peer. Ask learners what is meant by cloud computing, and how they use it. Explain the difference between private and public clouds and the benefits/drawbacks of each. Common misconception may be that it is only storage, but there is also cloud software. Where possible allow learners to use cloud software and storage. Ask learners to create a list of benefits and drawbacks of using cloud computing **(I)**; ask learners to share their answers with each other and to collect additional answers from other learners. Give learners an example scenario and ask them to justify whether cloud computing is appropriate in this situation. **(F)** Set learners up with threads/rope as different typologies: the learners are the nodes/server and the thread is the wired network connection. Rings or other objects can be used as the messages that need to be sent from one computer to another and transmitted along the threads.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Describe the characteristics, benefits and drawbacks, of different network topologies.	After each topology ask learners to consider what the benefits and drawbacks of that model will be. **(I)** Give learners scenarios and ask them to identify which topology would be most appropriate and to justify their choice. **(F)** Students should keep a glossary of networking terms as this topic is terminology-heavy and this should be updated each lesson.
2.1 Networks including the internet (hardware) **(KC3)**	Explain the differences between wired and wireless networks. Explain the benefits and drawbacks of both wired and wireless connections. Describe the purpose of hardware components that can support a LAN. Identify appropriate components to create a LAN. Describe the role and function of a router in a network.	Give learners a set of statements about the different wired and wireless communication methods, e.g. copped cable. Ask learners to work in pairs/small groups to identify which statements belong with which communication method. **(I)** Review each statement with learners. **(F)** Show learners an example network with labelled devices. Ask learners to work out what the purpose of the devices might be based on the diagram. Discuss the answers and tell learners the answers. Ask a network technician from the school (or another company) to come in and talk about the network devices in the school/company. Ask them to show the actual devices and how they are connected, etc. **Extension:** Allow learners to use the hardware components to connect devices and create a small LAN. **(I)** Give learners a scenario and ask them to design a network, identifying the different components that they would use. **(F)** Ask learners to justify their choices. Choose one learner to act as a router. Ask the other learners to act as nodes attached to the router, each with their own address. Connect the nodes using threads. Ask the router learner to perform the actions, e.g. take the packets of data from the node and forward them to the correct address (learner). **Extension:** Provide learners with a router and set of nodes; show learners how to set up the router to create a network. **(I)** Students should maintain a glossary of hardware components, giving their purpose/use.
2.1 Networks including the internet (Data transmission) **(KC3)**	Define collisions in data transmission and explain how Ethernet detects and avoids collisions.	Set learners up with a single line communication media, e.g. using a thread, with different learners connected to either end. Ask learners to send ‘data’ at the same time – ask them what happens, i.e. a collision. Explain how this means the data is lost. Explain how CSMA/CD detects this collision and how it manages it. Ask learners to repeat the exercise, this time using random time intervals to repeat the transmission.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Explain the difference between the internet and the WWW. Describe the hardware required to communicate over the internet. Explain the use of IP addresses in the transmission of data over the internet. Explain the benefits of a URL over an IP. Explain the role of a DNS in converting a URL to IP.	Ask learners what they do on the internet, and what they do on the world wide web. Challenge learners about what the differences actually are – it is likely they will not know. Ask learners to come up with a list of activities/actions they can do on the internet that does not involve the WWW. Explain how websites are stored on servers, and each time a webpage is requested, data is sent to the server that stores the website, the server executes the request and returns the required data. Ask learners to refer back to the lesson(s) on network hardware. Discuss the hardware and which are actually required, or optional, to access the internet (it should be that the majority are not required). Ask learners how they connect to the internet (e.g. home, school, on mobile phones) and explain the different modes. When discussing modems, refer back to sound (1.2 Multimedia: *Sound*) and that the modem performs the analogue to digital conversion (and vice versa). Ask learners to write down any IP addresses that they know, and any URLs that they know. Discuss the differences and how they are both actually addresses to websites, but one (URL) is more people-friendly. Use a telephone book (or other indexed book) to show learners how a DNS will find the IP for a URL, and use more than one book for when the answer is not in the first one. **Extension**: Ask learners to create a flowchart to show the process of converting a URL to IP. **(F)** Show learners the format for IPv4. Ask learners how many different addresses can be represented using IPv4 (assuming there are no reserved addresses). Ask them how many devices they think are ever connected to the internet at the same time. Explain the difference between public and private IPs, and static and dynamic IPs and discuss how this impacts the number of devices that can access the internet. Link to the need for more addresses so that more devices can be on the internet at the same time – introduce IPv6 and how this format allows for more addresses. **Extension:** If access permits, show learners how to ping URLs and ask them to explore what the data returned means. **(I)** Return to network diagrams that were drawn with topologies and hardware devices. Introduce subnetting and ask learners to add possible IPs onto the diagram for the router (or equivalent) and nodes. **(F)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/1 Specimen paper Q2		

Unit 3 Hardware

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
3.1 Computers and their components (primary and secondary storage) **(KC4)**	Explain the difference between primary and secondary storage. Identify items that are stored in secondary storage. Explain the difference(s) between RAM and ROM. Explain the difference(s) between SRAM and DRAM. Explain the difference(s) between PROM, EPROM and EEPROM.	Give learners access to an old computer and allow them to work in small groups to take it apart and try and identify the different components. This can be in done in reverse: give learners the components and challenge them to work out how to put it back together and get the computer working. **(I)** Discuss the difference between primary and secondary memory. Challenge the common misconception that secondary is only for backing up, not for storage of files. Secondary storage can be used to backup files, i.e. using removable storage, but it is not the main and/or only use. Give learners a list of files, documents, commands, elements of software, etc., that can be stored and ask learners to identify if they would be stored in primary and/or secondary (depending on how they are currently being used). **(F)** Ask learners to write down all the data that they store in secondary storage, and which device(s) this data is stored on, e.g. a PC, tablet, mobile phone etc. **(I)** Collate the learners' responses to give a clear indication of what is stored in secondary memory. **(F)** Explain the difference between RAM and ROM. Refer back to the list of data and ask them to identify if any of these would be stored in RAM/RAM. **(F)** Explain their purpose in a standard PC/laptop and then expand to a range of devices (e.g. laptops, mobile phones, televisions, games consoles, remote controlled car, etc.) and explain the differences in the purpose of RAM and ROM in each of these. Give learners a list of devices to investigate and identify what would be stored in RAM and ROM in each device. **(F)** Expand RAM into Static (SRAM) and Dynamic (DRAM). Ask learners to investigate the benefits and drawbacks of SRAM and DRAM, and why some of these are dependent on the actual tasks being performed. **(I)** Collate the answers from learners. **(F)** Expand ROM into PROM, EPROM and EEPROM, especially how these contradict the traditional definition of ROM that it cannot be changed. Ask learners to research examples of devices that make use of PROM, EPROM and/or EEPROM, what they are used for in these situations and why. **(I)** Collate the responses and share the answers for each type of ROM. **(F)** Students should maintain a glossary of hardware component terminology, adding new terms each lesson.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
3.1 Computers and their components (operation of hardware devices) **(KC4)**	Describe the principal operations of a range of hardware devices. Explain the purpose and use of buffers in a range of devices.	Give learners access to each of the hardware devices and allow them to take them apart and identify the components **(I)**. An alternative is one device can be shown to learners and they can observe it being taken apart. Explain to learners what a buffer is. Set up one learner as a sender, and one as a receiver. Provide the sender with a large number of messages written on small pieces of paper that have been screwed up into balls. The sender should send them (throw them) as fast as they can to the receiver, one at a time, and the receiver has to catch each one, read it out loud and then put it in a box before they can catch the next. They should find that the receiver cannot keep up and the messages end up on the floor. Introduce a 'buffer' – this could be another learner, or a box. Repeat, this time the sender sends the 'data' to the buffer, and the receiver collects from the buffer. The messages should not be lost. Ask learners to relate this use of a buffer to different scenarios e.g. printer buffers, transfer of data in a network. Put learners into groups and give each group a different device to investigate. Each group should produce a single-sided A4 sheet that shows: a) the components of a device, e.g. a cut-through model; b) annotations of components and their purpose; c) a flowchart that shows how the device works; and d) whether it might include a buffer and if so what its purpose is. **(I)** Groups should present their findings to the other groups, along with a copy of the info sheet. Link devices to previous topics, e.g. magnetic hard disk to secondary storage, microphone to multimedia (sound).
3.1 Computers and their components (monitoring and control systems) **(KC4)**	Describe the use of sensors. Identify appropriate sensors for a scenario. Explain the difference between a monitoring and control system. Describe the use and function of a monitoring and control system in a given situation.	Ask learners to identify all the different 'things' that they could measure, for example temperature. Collate the list and categorise them into the appropriate sensors – some learners may think there is a weight sensor when it is actually a pressure sensor. Take learners to view control and monitoring systems in practice, and/or to meet people who can explain what their monitoring and control systems to do. Ask learners to create a list of any monitoring and/or controls systems that they see and/or interact with during a week. Collate a list of these systems and ask learners to consider how they work. **Extension:** Provide learners with sensors that they can use to collect data, and set up both monitoring and control systems, e.g. through the use of Raspberry Pi https://www.raspberrypi.org/.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
3.2 Logic Gates and Logic Circuits **(KC4)**	Use the NOT, AND, OR, NAND, NOR and XOR logic gate symbols Understand and define the functions of : NOT, AND, OR, NAND, NOR and XOR (EOR) gates Construct the truth table for each of the logic gates Construct a logic circuit Construct a truth table Construct a logic expression	Provide learners with a list of statements that use AND, OR and NOT that require them to perform a function (most likely best integrated within IF statements, e.g. 'IF you are aged 17 AND wearing black socks then stand up'). Ask learners what the meaning of the AND, OR and NOT are in relation to each example, and which cases result in the action being performed (and not performed). Introduce the new operators NAND, NOR and XOR and give learners a series of these used in sentences. Give learners the truth tables for the six gates and show them how to complete a truth table for a statement. Give learners a series of logic circuits and ask them to complete the truth tables. **(F)** Show learners the gate symbols, and explain how they are combined to create logic circuits. This can be introduced using software, e.g. https://logic.ly/ **Extension:** Ask learners to work out what some of the other features/gates in the software do and how to use them. **(I)** Give learners a series of logic circuits and ask them to identify the circuits. Repeat, giving learners logic statements and ask them to create logic circuits. Some learners may benefit from working in pairs to begin with, and then working individually. **(F)** **Extension:** Ask learners to come up with the most difficult logic circuit they can draw, then they need to come up with the logic statement for the circuit. **(I)** Show learners some example problems and explain how to convert this into a logic expression. Give learners a series of problems and ask them to work in pairs to develop the logic statements, then develop the logic circuits and matching truth tables. **Extension:** Ask learners to come up with their own problem statements for other learners to develop logic statements and logic circuits from. **(I)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/1 Specimen paper Q4, 5		

Unit 4 Processor Fundamentals

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
4.1 Central Processing Unit (CPU) Architecture **(KC4)**	Describe the Von Neumann model for a computer system. Describe the purpose and role of each register in the Von Neumann model. Describe the purpose of and role of the components within the processor. Explain how the different ports allow connection to peripherals. Describe the stages of the Fetch-Execute cycle. Explain the purpose of interrupts. Describe how interrupts are handled in the F-E cycle.	Show learners a simulation of the Von Neumann model, e.g. the little man computer https://peterhigginson.co.uk/lmc/ Give learners a series of instructions to enter in, and ask learners to work in groups to work out the purpose of each component, e.g. register based on what happens when the instructions are run. **(I)** **Extension:** Ask learners to change the instructions/operands and to predict what will happen to the program. Then run the program and see if they were correct. Give learners the key terms and definitions (or a variety of statements detailing the function and purpose of each component) and ask them to match them. **(F)** Show learners how the F-E cycle runs. Set learners up as the different registers and components in the computer and get them to physically run the F-E cycle, and instructions, by passing data and instructions between each other. **(I)** Give learners the F-E cycle in register transfer notation with errors in and ask learners to find and correct the errors. **(F)** Give learners a series of different computer specifications and ask them which will run fastest, etc., and which components affect the performance. **(F)** Give learners a range of devices to connect to a computer using different ports to recognise the types of port and what they are used for. Ask learners to find out the benefits of each type of port and what it supports. Give learners scenarios and devices to be connected: learners should identify and justify a port to use for connection. A common misconception here is that USB is the device, e.g. a USB pen drive: learners should be made aware that USB is the connection and not the device. **(I)** Give learners a list of possible events that could cause an interrupt and ask them to categorise them, for example, into input/output, software, hardware, etc. **(F)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
		Ask learners to act out roles in the F-E cycle with one learner acting as the interrupt handler. When each F-E cycle ends, the learner acting as the interrupt register should be checked and the handler instigated if needed. **Extension:** Students can create a simulation of the F-E cycle, through animation software, or by writing a program that requires the instructions to be input. The simulation should show the contents of the registers at each stage and where data/instructions move to/from. Students should maintain a glossary of key terms for the components.
4.2 Assembly Language **(KC4)** **(KC2)** **(KC1)**	Explain the relationship between assembly language and machine code. Describe the stages of the assembly process for a two-pass assembler. Categorise assembly language instructions. Explain the different modes of addressing. Follow assembly language instructions to dry run a program.	Show learners an example of a high-level language program, the equivalent in assembly language and then in machine code. Discuss how all the programs are the same and how the assembly language is made up of opcodes and operands. Walk through the stages of the two-pass assembler, or develop an animation showing each step. Give learners some simple programs and ask them to perform the same process. Compare learners' results for the processes and ask them to identify the differences and what any errors were. **(F)** Give learners assembly language programs and ask them to write out what each instruction does using the assembly language table but with application to the operand in the code. **(I)** Use an assembly language simulator to run assembly language programs, e.g. https://peterhigginson.co.uk/lmc/ **(I)** Give learners assembly language programs and trace tables to complete – ask learners to work in pairs to begin with, one following the code and the other updating the table to keep track of where they are. **(F)(I)** **Extension:** Give learners assembly language programs that have an error and ask them to trace the programs to find out what the problem is and what the solution should be. **(I)** Use boxes with instructions like a treasure map to represent modes of addressing. For example, the box might have the address of another box to open. Give learners an instruction and ask them to execute the instructions by opening the box(es) where appropriate and performing the actions. Give learners a list of assembly language instructions and ask them to group them into the five given groups (immediate, direct, indirect, indexed, relative). **(F)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
4.3 Bit manipulation **(KC4)** **(KC2)** **(KC5)**	Perform shifts on a binary number. Explain the impact of a shift on a binary number. Use bit manipulation to check values in registers.	Show learners how to perform a shift, then ask them to work out the effect a single shift has on the number. Repeat with a 2-place shift, and left and right, etc., so learners are working out the purpose for themselves. **(I)** Ask learners what the problem(s) with shifts are, e.g. bits are lost, and explain how these could be mitigated. Show learners the effect of bit manipulation operations. Give learners examples of where and why these would be used for example relating to scenarios to check the values of bits, and to change the values of bits. Relate back to monitoring and control, recap fundamentals of sensors and characteristics of these systems. Show learners how registers can be used to check the data received and perform actions depending on their content. Give learners a scenario and ask them to work in pairs to follow and examine assembly language programs to perform monitoring and control tasks. **(I)** **Extension:** Students can create assembly language programs to test bits and perform monitoring and control activities.
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)**		

Unit 5 System Software

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
5.1 Operating System **(KC4)**	Explain why a computer system requires an Operating System. Explain the key management tasks carried out by the Operating System. Explain the need for utility software. Describe the purpose and function of typical utility software. Explain the purpose of program libraries and the benefits of using a library (including DLL).	Ask learners to list what the Operating System they use allows them to do. Ask them to consider a computer without an operating system – how would they perform tasks? Introduce each management task of an Operating System. Relate to different types of computers and how it might be the same, or how the task may differ, e.g. tablets, mobile phones, embedded systems, etc. Give learners a list of actions that may be performed and ask learners to categorise them into the management activity they relate to. **(I)** Ask learners to write down all the useful software that comes with their operating system i.e. the utility software. **(I)** Give learners access to a program library for the program language used. Show learners how to import the library and use the procedures within it. Where possible, let learners use both static and dynamic libraries to experience the differences. **(I)** Give learners an example scenario, e.g. a person who could choose to use a library or not, and ask them to argue for a library and against. This could extend to dynamic or static libraries. **(F) (I)**
5.2 Language Translators **(KC4)** **(KC2)**	Identify the purpose of an assembler, compiler and interpreter. Explain the benefits of using a compiler and/or interpreter in a given situation.	Give learners some instructions that are encoded (e.g. using a cipher), and ask them, in pairs, to act as a compiler and an interpreter. The compiler should translate all the lines and then do what they say; the interpreter should translate one line and then run it. Ask learners to debate compiler vs interpreter for a given scenario, e.g. a person who is writing a program. Students should argue why each should be used – there may not be an actual definite answer, but it is the reasons behind the arguments that are important. **(F) (I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Describe the features found in an IDE.	Ask learners to open the compiler/interpreter IDE they use for the programming language used, and to write down all of the tools and features it includes. Give them access to a different compiler/interpreter and ask them to go through the list and identify which it also has, and if there are any different/extra features. **(I)** Give learners the categories coding, initial error detection, presentation and debugging – ask learners to put the features/tools into each of categories. Collate these from the learners and add any extra they are missing. **(F)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)**		

Unit 6 Security, Privacy and Data Integrity

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
6.1 Data Security **(KC3)** **(KC1)** **(KC5)**	Explain the difference between security, integrity and privacy of data. Describe the threats to data and computer systems. Explain how threats can be prevented or restricted. Describe methods to secure data.	Collect recent news articles about companies that have encountered major issues, e.g. through a lack of security, privacy of data. Ask learners to discuss what they did wrong and what they should have done, or what they should do next. Ask learners to list the different ways they can keep their data, and computer secure. Give each learner (or pair of learners) a security measure to research. Ask them to create a presentation about what the measure is, how it works, etc., in technical detail. Learners can present their findings to each other. **(F) (I)** Put learners into groups and give each learner a different scenario. Ask them to create a report identifying the potential threats to the data and computer systems in the scenario, and what they should do to prevent/limit these threats. **(F) (I)** **Extension:** Students can program an encryption algorithm that both encrypts a message and decrypts using a common encryption method. **(I)**
6.2 Data integrity **(KC3)** **(KC1)** **(KC5)**	Describe different validation routines. Explain how verification can be used to make sure data is the same as the original. Explain how data can be verified during data entry and transfer.	Ask learners to complete some online forms that include verification and/or validation. Ask them what restrictions there were. Link these to whether they are examples of verification or validation. Give learners a series of validation descriptions and ask them to categorise them into one of the methods, for example some may be examples of code. **(F)** Ask learners for examples of when they have had to verify data, e.g. new passwords, confirming amounts, etc., and create a list of all the different places this has been encountered. **Extension:** Give learners a program that requires a range of inputs and ask them to program validation and verification routines. **(I)** Discuss what problems could occur during data transfer. Show learners how to add a parity bit. Get learners to 'send' data to each other using parity. Ask them how they could find the error that occurred.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
		With a parity byte you cannot find it – so introduce parity blocks and how an error can be located and therefore corrected. Ask learners what happens when 2, 3, 4, etc., errors occur instead of just 1 bit, and how parity may not identify that an error has occurred, and the location can no longer be found. Introduce checksums and ask learners to perform checksums on data.
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/1 Specimen paper Q4		

Unit 7 Ethics and Ownership

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
7.1 Ethics and Ownership (ethics and copyright) **(KC3)** **(KC1)**	Explain the need for ethics and to act ethically. Discuss the impact of acting ethically and unethically. Identify ways a person can act ethically and/or unethically in a given situation. Describe the key features of a range of software licences.	Give learners a range of scenarios and ask them to identify the ethical decisions that could be made. Organise debates as to whether an action, or scenario, is ethical or unethical with learners fighting for both sides. Give learners the IEEE ethics and a scenario, ask them to identify one way that a specific person could act in the ethical interest of each of the IEEE categories. Ask learners to discuss potential consequences on not acting ethically, or unethically. **(F) (I)** Give learners some example software with the copyright licences included. Ask them to summarise what the licence does, and does not, allow them to do. Give learners a scenario and ask them to recommend a licence, with justification. Ask some learners to fight for one licence and other learners a different one. Explain that there is not necessarily a right answer, it is about the justification. **(F)** Give learners features of different licences and ask them to categorise them to the appropriate type of licence. **(F) (I)** Students should maintain a glossary of the different licences.
7.1 Ethics and Ownership (Artificial Intelligence) **(KC3)** **(KC1)**	Identify the need for Artificial Intelligence (AI). Discuss the benefits and drawbacks of AI.	Show learners a clip from a film or television programme where AI is evident. Discuss how far from reality this is, and what future developments are needed before this could become a reality. Give learners an example of where AI is being developed (or could be) and ask learners to identify the potential positive and negative impacts this could have. **(F)** Put learners into two teams, one for AI and one against. Hold a debate for and against the development of AI. **(F) (I)** **Extension:** Students can explore how computer programs can 'learn' and adapt to new situations, for example by looking at ant algorithms.

Past and specimen papers
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/1 Specimen paper Q4e

Unit 8 Databases

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
8.1 Database Concepts **(KC5)** **(KC1)**	Explain the limitations of a file-based approach. Describe the features of a relational database that addresses the limitations of a file-based approach. Create entity-relationship (E-R) diagrams to document a database design. Describe the normalisation process of a database. Create a normalised database design for a given database description.	Give learners a file-based database and ask them to perform some actions, e.g. change data, retrieve data. Ask them what the problems were, e.g. they had to change the same data multiple times, data had to be accessed across several files. List the drawbacks of a file-based approach and explain how a relational database will solve this problem. Give learners a relational database that has already been set up and populated, ask them to perform tasks as with the file-based approach and compare the two systems. **(I)** Give learners an example database and ask them to apply the terms, e.g. identify the entity, a table, record, etc. Give learners database terms with definitions and ask them to match the definitions to the terms. Ask learners to describe the relationships between a series of tables, both from descriptions and then from E-R diagrams. Set up a spreadsheet with the fields for a database in 0NF with some sample data. Ask learners to work together to normalise the database – using the spreadsheet to manipulate the field names into the new tables. Give learners an example database and ask them to identify which NF it is in and to justify their choice. **(F)** Provide learners with a database scenario and ask them to identify the fields required, and then to design a normalised database. **(F) (I)** Students can provide a guide to normalisation, using an example database, to explain each step of the normalisation process. **(F)** Students should maintain a glossary of key database terms.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
8.2 Database Management System (DBMS) **(KC5)** **(KC1)**	Explain how a DBMS addresses the limitations of a file-based approach. Describe the features and software tools of a DBMS.	Give learners experience of one or more DBMSs. Ask them to identify the features/tools within it. Collate a list of these features. Show learners a database that does not have a DBMS; for example, one that uses SQL only, and ask learners to compare them giving the benefits and drawbacks of each. **(F) (I)** Show learners how to set up a database using the DBMS and a range of features. Give them a scenario and ask them to set up a database making use of the different features. Ask learners to describe the features they used and how they address the limitations of a file-based approach. Give learners a scenario where a range of people with different requirements access a database. Ask learners to identify the data that each person or group of people needs to access, and relate this to the different security and system views that they should have access to. Students should maintain a glossary of key database terms.
8.3 Data Definition Language (DDL) and Data Manipulation Language (DML) **(KC5)** **(KC2)** **(KC1)**	Follow DDL and DML commands written in SQL. Write SQL scripts to perform DDL and DML tasks.	Introduce each SQL statement one at a time and give learners a range of exercises using that statement. **(F)** Ask learners to recreate a database they have already made using SQL only. **(F) (I)** Give learners a series of SQL scripts with errors in them. Ask learners to correct the statements and run them on a database. Give learners a new database scenario. Ask them to create a normalise model for the database and then set it up using SQL scripts. **(F) (I)** Ask learners to give each other challenges, e.g. come up with a query that they want results for, and the other learner must complete the challenge. **(I)** Give learners 'fill the gaps' SQL statements and scripts that are incomplete; learners should fill in the missing statements. Give learners SQL scripts that are out of order and ask them to put them in the correct order. Students should produce an SQL glossary of command terms and their meanings.

Past and specimen papers
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/1 Specimen paper Q3

Unit 9 Algorithm Design and Problem Solving

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
9.1 Computational Thinking Skills **(KC1)**	Explain the purpose of and need for abstraction. Create an abstract model of a system. Explain the purpose of and need for decomposition. Decompose a problem into its sub-problems.	Show learners a series of computer programs, e.g. different games, and ask them to explain the difference between the real world and the representations within the game, e.g. the detail, elements that are missing, what has been included. Discuss why some elements have been removed, and how the real world features have been represented in the computer system. Ask learners what problems they could foresee happening if they tried to create an exact replica of the real world in a computer game – e.g. time constraints, memory limitations, processor requirements. Give learners a problem, e.g. a computer game, and ask them to create an abstract model by identifying what details they would keep and remove, and how they would represent parts of the system. **(I)** Give learners the description of a computer game they need to create. Ask them to consider where to start and what they will do first. Discuss the issues with a large problem and what can be done to make this simpler. Show learners how to decompose a problem and use a structure diagram to represent the component parts. Ask learners to decompose the same problem and then identify where to start. Discuss how this decomposed problem could now be split among different people, and each component can be a program module and interpedently created. **(F) (I)**
9.2 Algorithms **(KC1)** **(KC2)**	Select appropriate identifier names. Write programs in pseudocode using input, process and output. Write pseudocode using assignment, sequence, selection and repetition (including logic statements).	Give learners a program with multiple variables in that are all very similar in name that are also meaningless, e.g. zz1 zzz1 zzz1. Ask learners what problems could happen with these identifiers. Discuss what makes a good identifier. Give learners a program and ask them to create an identifier table. Give learners algorithms in pseudocode to follow. These could start with instructions of actions for them to take, such as moving around the room, and then progress to variables and processes on these. Give learners pseudocode algorithms with errors and ask learners to trace the algorithms and correct them. **(I)** Give learners pseudocode algorithms and ask them to identify where there are examples of assignment, sequence, selection and repetition. Introduce selection using a series of statements for learners to read and follow e.g. if the time is 11:00 and you are 17 years old then clap your hands, etc. These can be extended to multiple criteria and actions for learners to work

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Write pseudocode from a structured English description and from flowchart. Explain how stepwise refinement can be used to express an algorithm to a level of detail from which the task may be programming.	out what to do. Boxes can be given identifiers with data written on paper inside, then learners can follow selection statements by opening the box and checking the value within it. Give learners short programs to write pseudocode solutions to, these can increase in complexity each time. Students can design the programs in pseudocode and then write the program in their chosen programming language. Give learners a program in pseudocode and program code and ask them to identify the differences, i.e. language specific terms are not used, but that there is no set pseudocode. Give learners flowcharts to follow that make them perform actions. Ask learners to then convert these flowcharts into pseudocode. **(I) (F)** Ask learners to design the solution to a program in pseudocode, then to give this pseudocode for someone else to follow and create a program from. **(F) (I)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/2 Specimen paper Q5, 6, 7		

Unit 10 Data Types and Structures

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
10.1 Data Types and Records **(KC1)** **(KC2)** **(KC5)**	Select and use appropriate data types for a problem solution. Use a record structure to hold a set of different data types under one identifier.	Give learners example data and ask them to identify the most appropriate data type to use. Give learners programs where inappropriate data types were chosen and ask them to identify why they are inappropriate and to suggest one that is more appropriate. **(F)** Present learners with a pre-created record structure and ask them to create new records, store data in them and access the data. **(I)** Give learners a scenario and ask them to design a record structure to store the data in. Ask learners to create the structure, store examples of data in it and extract the data from it. **(I)** Students should maintain a pseudocode glossary of the key terms they can use to define the different structures.
10.2 Arrays **(KC1)** **(KC2)** **(KC5)**	Use the technical terms associated with arrays. Select a suitable data structure (1D or 2D array) to use for a given task. Write pseudocode for 1D and 2D arrays. Write pseudocode to process array data.	Give learners a grid on paper with indexes above each space. Explain how to put data in an index, get data from an index, etc., then learners can perform each instruction on the grid, writing in the values, etc. This can be repeated with a 2D grid. **(F)** Provide learners with small pieces of paper with numbers on to shuffle and place in the 1D/2D array grid facing down. Learners can use the cards to perform a bubble sort and linear search, turning the cards over on each stage. **(I)** Give learners a pseudocode algorithm for a bubble sort and linear search and ask them to trace the algorithm. Repeat with algorithms that include errors and ask learners to trace the algorithms to find, and correct, the errors. Give learners incomplete pseudocode algorithms for bubble sort and linear search and ask them to complete the algorithms. **(F) (I)** Discuss efficiency in a linear search and bubble sort. Present learners with inefficient algorithms and ask them to work out how to increase the efficiency. **(I)** **Extension:** Ask learners to investigate other searching and sorting algorithms that are more efficient than linear and bubble. Ask them to work out how they work and why they are more efficient. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
10.3 Files **(KC1)** **(KC2)** **(KC5)**	Explain why files are needed. Write pseudocode to handle text files that consist of one or more lines.	Discuss the need for files and what would happen to data if it was not stored in a file. Give learners pseudocode for using a file with error(s) in them and ask learners to find the error(s). **(I)** Present learners with a scenario that needs file storage and ask learners to write pseudocode for the program. **(F) (I)** Give learners pseudocode for a program that accesses a file that includes errors, ask learners to trace, find and correct the errors. **(F) (I)** **Extension:** Ask learners to research different methods of file organisation and access, and to find out how the pseudocode would have to change to take this into account. **(I)** **Extension:** Ask learners to work out how to write program code in a specific language to access files and to convert the pseudocode to program code. **(I)**
10.4 Introduction to Abstract Data Types (ADT) **(KC1)** **(KC2)** **(KC5)**	Explain that an ADT is a collection of data and a set of operations on those data. Explain how a stack, queue and linked list are examples of ADTs. Use a stack, queue and linked list to store data. Describe how a queue, stack and linked list can be implemented using arrays.	Act out stacks and queues with learners: learners form a line – the stack/queue – and then other learners are added and/or removed according to the instructions being run. Give learners a grid on paper to act as the stack/queue and data items on individual pieces of paper. Learners add data to the queue by putting the pieces of paper onto the grid, and remove data from the stack/queue by them from the grid. **(I) (F)** Act out linked lists with learners. Each learner will need a number, and a number that they point to. The learners can then be added to the linked list and each learner changes their pointer to point to the next learner in the list. Reinforce the use of null pointers with the final element pointing to null. Give learners a 2D table on paper to act as a 2D array, with column headings, e.g. index, data, pointer. Learners are to follow a set of instructions to populate the linked list by filling in the table. This can be extended by removing items and learners having to update the table each time. **(F) (I)** **Extension:** Students could begin to develop algorithms for adding, removing and editing the data in the three structures. These could be in the form of structured English instructions, flowcharts and/or pseudocode. **(I)**

Past and specimen papers
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/2 Specimen paper Q1, 3, 5, 7

Unit 11 Programming

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
11.1 Programming Basics **(KC1)** **(KC2)** **(KC5)**	Write pseudocode from a given design presented as either a program flowchart or structured English. Write pseudocode 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 input from the keyboard and output to the console	Introduce variables and constants using boxes that have the data within them (e.g. on paper). Give learners a series of pseudocode statements that they run by getting the data from the boxes and/or changing the data in the boxes and/or reading out the data from within the boxes. Provide learners with pseudocode that uses arithmetic and logical operators, variables and constants that includes errors; learners should trace the algorithms and identify the errors. **(F) (I)** Give learners access to built-in functions and library routines in the chosen programming language. Discuss what the libraries allow them to do that they could not before. Show them what happens if they try and use the routines without importing them. Give learners a list of the different arithmetic and logical operators and ask them to identify their meaning. Give learners pseudocode algorithms with missing operators and ask them to complete the operators. **(F)** Learners should have a glossary of key programming terms and the arithmetic and logical operators should be added. You should refer to the Pseudocode Guide on the School Support Hub www.cambridgeinternational.org/support while teaching this topic.
11.2 Constructs **(KC1)** **(KC2)** **(KC5)**	Use pseudocode to write: • an IF structure including ELSE and nested IF statements • a CASE statement • a count-controlled loop	Present learners with a range of IF statements to execute practically, by performing actions, calculations, etc. **(F) (I)** Give learners a program with an IF statement in and ask them to turn it into a CASE statement. Repeat with examples that cannot be converted and ask learners to explain why it cannot be represented in the alternative. **(I) (F)** Provide learners with a CASE statement and ask them to turn it into an IF statement. **(I) (F)** Give learners example selection statements with errors in and ask them to debug the code and correct the errors. **(F) (I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	• a post-condition loop • a pre-condition loop Justify why one loop structure may be better suited to solve a problem than the others.	Present learners with a range of iteration statements to execute practically, e.g. to continually do something until a condition is met. This can be repeated with calculations, etc., that are performed practically, taking one line of code at a time. Give learners a program using one type of loop and ask them to convert it into another type, e.g. a pre-condition to a post-condition. Ask learners to identify what types of loop cannot be converted into other types, e.g. not all post/pre-condition loops can be turned into count-controlled loops. Provide learners with pseudocode algorithms with loops in that are incorrect, e.g. they loop 11 times instead of 10 times. Ask learners to dry run the algorithms and correct the errors. **(F) (I)** Give learners pseudocode algorithms with loops in and ask them to write them in your chosen programming language. **(F) (I)** Describe programs that require loops, and ask learners to identify and justify the most appropriate loop to use. **(F) (I)** Learners should create a glossary of key programming terms and ask selection and iteration operators.
11.3 Structured Programming **(KC1)** **(KC2)** **(KC5)**	Define and use a procedure and a function. Explain where in the constructor of an algorithm it would be appropriate to use a procedure or a function. Use parameters in a procedure and a function.	Give learners some programs (in pseudocode or a programming language) that include procedures and procedure calls. Ask learners to work in pairs to trace the algorithms, following each instruction to work out what the procedures do and what they are. Ask learners to explain their findings. **(F) (I)** Show learners an algorithm with a section of code that is repeated multiple times, e.g. outputting the contents of an array. Ask learners what the drawbacks of repeated code are and explain how procedures can reduce this repeated code. **(F)** Show learners some built-in functions that they will already have made use of in their language e.g. finding the length of a string. Discuss what is different between a procedure and what the function does. Give learners a description of functions to implement and then make use of in their program. **(I)** Use boxes to act as variables and parameters. Show learners what happens to the values in the variables, and parameters when an algorithm is run that includes procedures taking parameters by value and then by reference.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Use the terminology associated with procedures and functions. Write efficient pseudocode.	Ask learners to explain to each other what by reference and by value means and when these should be used. Give learners algorithms that include procedures and/or functions that take values by reference and/or by value that are incorrect and ask them to trace them and make corrections. **(F) (I)** Give learners pre-written subroutines to implement and make use of in a program. **(I)** Show learners a structure diagram (or make use of one they created earlier) and discuss how this design can be used to identify the procedures/functions. Give learners scenarios and ask them whether procedures or functions should be used. Ask learners to justify their choices.
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/2 Specimen paper Q1c, 4		

Unit 12 Software Development

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
12.1 Program Development Life cycle **(KC1)** **(KC2)**	Explain the purpose of a development life cycle. Explain the need for different development life cycles depending on the program being developed. Describe the principles, benefits and drawbacks of each type of life cycle. Describe the analysis, design, coding, testing and maintenance stages in the program development life cycle.	Present learners with two scenarios, one where a development team are following a plan/schedule based on a development life cycle, and another where they are all working on the program without any plans, etc. Ask learners to discuss the benefits and drawbacks of each approach. **(F)** Describe each stage of a development life cycle one at a time, showing learners the items developed in that stage; give learners a small program and after each stage ask them to follow the same processes for their project. **(I)** Give learners the different life cycle stages and activities on individual cards; ask them to put the stages in order and to put the activities with their corresponding stage. Repeat for the different life cycles. **(F)** Provide learners with a series of benefits and drawbacks for the different development life cycles and ask them to identify which cycle each belongs to. **(F)** Give learners a series of scenarios of projects that need to be written and ask learners to work in groups to identify which development life cycle should be used and to justify their choice. **(I) (F)** Give learners a scenario program to develop as a group (this could be as a whole class, or in smaller groups). Ask learners to select a life cycle to follow and to justify their decision, and then to follow the life cycle to create the program as a group – limit the scope of the project to allow them to use all sections of the life cycle. **(F) (I)** **Extension:** Ask learners to research other typical life cycles and identify the common themes between them, and the differences. **(I)**
12.2 Program Design **(KC1)** **(KC2)** **(KC5)**	Use a structure chart to decompose a problem into sub-tasks and express the parameters passed between the various modules/procedures/	Show learners a structure chart and ask them to explain each part of the diagram. Give learners a structure chart and ask them to create the program that it shows. **(F)** Give learners a structure chart without the parameters and ask them to complete the diagram. **(F)** Give learners a structure chart and pseudocode for the same program and ask them to identify the differences. **(F)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	functions which are part of the algorithm design. Create and use a state-transition diagram to document an algorithm.	Show learners a device, such as a safe with a programmable number. Ask learners to work in groups to identify all the different states the device can be in. Ask learners to then identify how the device moves from one state to another. Show learners how to turn this into a state-transition diagram. **(F) (I)** Give learners a state-transition diagram and ask them to convert it into a narrative of the different states and the movement between those states. Ask learners to create a state-transition diagram for a scenario/program. **(F) (I)** Show learners a computer program, such as a game with characters and objects. Ask learners to identify the different objects, and then states these objects can be in. Ask them to draw state-transition diagrams for the objects. **(F) (I)** **Extension:** Ask learners to turn one of their state-transition diagrams into a program. **(I)**
12.3 Program Testing and maintenance **(KC1)** **(KC2)**	Explain how faults in programs can be exposed and avoided. Locate and identify the different types of errors. Correct identified errors. Use different methods of testing and select appropriate data for each method. Explain the need for a test strategy and	Give learners a program that does not work, and ask them to find the error, using whatever methods they want. Collate the methods used by the different learners into a list of ways to find errors. Split this into ways that they worked out there was as error, and how they found out where and what the error actually was. Show learners a series of programs each with one or more types of error in. Ask learners to read the program and identify the type(s) of error within it, e.g. if there is a syntax error, logic error etc. and then where this is. **(I)** Give learners a program with errors in it (or ask them to write a new one) and then perform each type of testing and document their testing, i.e. dry run, walkthrough, white-box, black-box, and so on. **(I)** Show a series of algorithms and test data and ask learners to identify the type of test data each is for. Repeat but by giving the type of test data and asking for examples of data that could be used. **(F) (I)** Ask learners to complete test plans for the programs they write while learning new constructs. **(F)** Give learners programs that do not work, either at all or as required. Ask learners to find and correct the errors using a range of testing methods. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	test plan, and their likely contents. Choose appropriate data for a test plan. Explain the need for continuing maintenance of a system and the differences between each type of maintenance. Analyse an existing program and make amendments to enhance functionality.	Use case studies of companies that have had problems with systems after implementation which required maintenance (examples of adaptive/corrective/perfective maintenance). Discuss the need for this maintenance and the potential consequences of not undertaking it.
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/2 Specimen paper Q2, 6bfile		

Unit 13 Data Representation

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
13.1 User-defined data types **(KC1)** **(KC2)** **(KC5)**	Explain why user-defined data types are necessary. Define and use non-composite data types. Define and use composite data types. Choose and design an appropriate user-defined data type for a given problem.	Give learners examples of the different pre-determined data types for your chosen language. Discuss the limitations of these and ask learners how they can get around this problem (e.g. multiple variables/arrays, use classes, etc.). Explain the difference between composite and non-composite data types in pseudocode and your chosen language. Discuss the need for both types. Give learners a scenario and ask them to decide whether composite/non-composite data types are required for each data item or series of items. Ask them to justify their choice. **(F) (I)** Give learners program code that already has a range of user-defined data types written, and ask them to use these to create a given program. **(I)** Present learners with a scenario and ask them to program a user-defined data type for the program. Ask learners to justify the data type they created. **(F) (I)** Give learners examples of data stored in a data type, and declarations of data types. Ask learners to identify which are non-composite and which are composite. Extend this to the different types within non-composite and composite. **(F)** Provide learners with programs that define and use a range of data types including user-defined data types, and ask them to correct the errors. **(F) (I)** **Extension:** Introduce learners to a second programming language that has different data types and allows for different definitions of user-defined types. Ask learners to compare the languages and these features. **(I)**
13.2 File organisation and access **(KC1)** **(KC2)** **(KC5)**	Describe the different methods of file organisation. Describe the different methods of file access.	Discuss example programs and the need to store/access data stored externally to the file. Discuss the consequences of not storing the data accurately. Put example records/data for a file onto individual pieces of paper. Explain to learners how each method of file organisation works and ask learners to physically implement this structure using the data by placing the papers in the correct order as that method would store it. This could also be done using a spreadsheet with the file locations being the row numbers.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Select an appropriate method of file organisation and file access for a given problem. Describe and use hashing algorithms.	Describe common hashing algorithms and ask learners to work out the record location for a piece of data or record using one of these algorithms. Show learners a series of data stored as a random file but that is full (or nearly full). Ask learners to work out the file location using the hashing algorithm and ask them what problem occurs – i.e. clashes. Ask learners to work out what could happen when there is a clash, and how this solution would affect the access of the data. **(I)** Give learners example files with large quantities of data stored in a specific organisation method (for example, using a spreadsheet). Ask learners to use the different methods of file access to find the required data. Discuss the benefits and drawbacks of each method. **(F)** Give pairs of learners a scenario and ask them to decide which file organisation method should be used for storage and for access. Students should explain their choice with justification. **(F)** Provide learners with a range of scenarios or situations (such as number of files, type of data being stored, etc.) and ask them to group them by the most appropriate method of file organisation, and then access based upon this. **(I)** Show learners how to write code to perform each of the different methods of file organisation and ask them to implement them in a range of programs. Extend this to use different methods of file access to find and read data from their files. Give learners program code that uses different methods of file access and organisation that includes a number of errors. Ask learners to find and correct the errors. **(I)**
13.3 Floating-point numbers, representation and manipulation **(KC5)**	Describe the format of binary floating-point real numbers. Convert binary floating-point read numbers into denary and vice versa.	Recap binary conversion, two's complement and addition and subtraction. Ask learners what other types of number exist that may need to be stored in a computer, i.e. decimals. Show learners how a fixed-point binary number is calculated. Give learners numbers to convert into fixed-point and vice versa. Discuss the limitations of fixed-point, i.e. the binary point is always in the same place, so the range of numbers is limited. Give learners example numbers to store in set formats and ask learners to work out why they cannot be represented i.e. there are insufficient bits before/after the binary point. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Normalise floating-point numbers. Explain the consequences of a binary representation only being an approximation to the real number it represents (in certain cases). Explain that binary representations can give rise to rounding errors.	Show learners how to convert a fixed-point number into floating-point and vice versa. Give learners a range of denary numbers to convert into floating-point and then vice versa. Start with positive mantissa and exponents, then expand this to include negative mantissas, negative exponents and then both as negative. **(F)** Show learners a range of floating-point numbers and ask them to identify the mantissa and exponent for each. **(F)** Ask learners to work the largest and smallest numbers that can be represented in that format. Repeat with a different number of bits in the mantissa and exponent. Ask learners to work out what difference the size of mantissa and exponent has on the numbers that can be represented. **(I)** Show learners how some numbers cannot be exactly represented. Show learners what happens when this occurs in a program (or on a calculator). Discuss what potential problems this could cause in programs.
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q1		

Unit 14 Communication and Internet Technologies

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
14.1 Protocols **(KC3)**	Explain why a protocol is essential for communication between computers. Describe protocol implements as a stack, with each layer having its own functionality. Describe the TCP/IP protocol suite. Describe the purpose of the protocols HTTP, FTP, POP3, IMAP, SMTP, BitTorrent.	Ask learners to write a message in binary to another learners. This can be done by writing the binary, or using lights to flash binary values. Students should not speak to each other prior to, or during, the sending of the message. Ask learners what problems they encountered – i.e. not knowing what the letters represented, etc. Ask learners what they need to do before sending – link to protocols and their purpose and what they identify. Students can then agree protocols and resend the message, which should be more successful this time. **(I)** Give learners example protocols and functions of the different layers and ask learners to identify the layer they belong in. **(F)** Relate protocol layers to a physical example, e.g. a series of boxes on top of each other, where one can be removed, changed and then replaced independently. Create a crossword of protocols with the descriptions as the clues for learners to complete. Give learners a series of scenarios and ask them to identify the most appropriate protocol(s) to use, and to justify their choice. **(F) (I)**
14.2 Circuit switching, packet switching **(KC3)**	Explain the purpose, benefits and drawbacks of circuit switching and packet switching. Justify the use of packet and/or circuit switching in a scenario.	Show learners how data is split into packets and the contents of a packet. Give learners some data to split into packets (binary data or textual). Ask learners to split the data into packets and complete a packet header for each. Act out physical circuit and packet switching using learners as nodes and setting up connections between them, e.g. using string. One learner acts as the sender, and another as the receiver. When performing circuit switching, the sender identifies which nodes the message will goes through, then sends it one packet at a time. When performing packet switching, the sender sends each packet one at a time, then each node decides which way to send it.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
		Give learners statements, benefits and drawbacks about circuit and packet switching, and ask them to identify which they relate to. **(F)** Put learners into pairs and give each pair a scenario. Ask them to decide whether packet or circuit switching is most appropriate in this situation and to justify their choice. Ask them to describe how their chosen method will transmit the data. **(F) (I)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q2		

Unit 15 Hardware and Virtual Machines

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
15.1 Processors, Parallel Processing and Virtual Machines **(KC4)**	Describe Reduced Instruction Set Computers (RISC) and Complex Instruction Set Computers (CISC) processors. Explain the importance and use of pipelining and registers in RISC processors. Describe the four basic computer architectures (SISD, SIMD, MISD, MIMD). Describe the characteristics of massively parallel computers. Describe the concept, benefits and limitations of a virtual machine.	Give learners example instructions from a RISC and a CISC processor. Ask learners to identify the differences. Ask learners to work out what the benefits and drawbacks of each could be. Discuss learners' answers and lead them to creating a list of benefits and drawbacks. **(F)** Give learners a series of statements about RISC and/or CISC and ask learners to identify which relate to RISC and which to CISC. **(F)** **Extension:** Ask learners to research how some calculations can be performed through combinations of simpler calculations, for example how division can be performed through subtraction. **(I)** Provide learners with a timeline table and the stages of the Fetch-execute cycle for a number of different processes. Ask them to complete the table by showing which process is being fetched/decoded etc. in each time period. Give learners a table with the number of instructions and data as the row and column headings. Ask learners to put the architectures into the correct position according to the headings. **(F)** Give learners examples of instructions that are performed using each of the four architectures. Show learners a range of applications for each architecture and discuss the differences in application requirements for each architecture. Discuss the need for increasing processor power and how instead of increasing the processing power of one computer, multiple systems can be used in combination. Ask learners if they have ever used emulator software – popularly used to play console games on a PC. Relate to the virtual machine concept. Discuss the specific hardware required for the games console and how this is met by the PC instead. Ask learners for the benefits and drawbacks. Give learners a list of statements, some true and others false, about virtual machines. Ask learners to identify which are true and which are false. Discuss the answers. **(I)** **Extension:** Ask learners to research distributed systems and how these compare to parallel computers. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
15.2 Boolean Algebra and Logic Circuits **(KC4)** **(KC1)** **(KC5)**	Produce truth tables for logic circuits including half adders and full adders. Describe the function and create a truth table for a flip-flop (SR, JK). Use Boolean algebra to manipulate Boolean expressions. Describe the use of, and use a Karnaugh map (K-map).	Recap logic circuits from Unit 1. Give learners a series of logic statements and ask learners to produce logic circuits and truth tables from the statements. **(F)** Demonstrate the half adder and full adder. Ask learners to walk through the diagrams keeping track of the values at each step. Ask learners to work out why they are called a half adder and full adder. **(I)** Demonstrate the flip-flop circuit and ask learners what the purpose of its existence is. Link to the name and why it is called a flip-flop. Give learners incomplete adder and flip-flop circuits. Ask learners to complete the diagrams. **Extension:** Ask learners to draw the adder and flip-flop circuits using only NAND gates. **(I)** Show learners two logic statements that have identical outputs and ask learners to work out what the difference is, i.e. they use a different combination of gates. Discuss how many ways one circuit can be drawn and which is the best – i.e. the most efficient – and why. Show learners how to use a K-map to simplify a logic statement. Give learners truth tables to create and complete K-maps from. Gradually increase the difficulty in the number of inputs. **(I) (F)** Explain how K-maps are only one way of simplifying a logic statement. Show learners the Boolean algebra rules one at a time. Slowly increase the complexity by asking learners to use two rules with one statement, then increase again. Give learners incomplete Boolean algebraic expressions with missing components and ask learners to complete the formulae. **(F)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q3		

Unit 16 System Software

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
16.1 Purposes of an Operating System (OS) **(KC4)**	Explain 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. Describe how processes are managed by the OS. Describe the use of virtual memory, paging and segmentation for memory management.	Recap on OS functions from AS and the management of memory, hardware etc. Discuss rules the OS can implement to maximise the use of these resources. Give learners a list of events that they would regularly perform on a computer, e.g. opening a piece of software, saving work, listening to music, switching between windows, etc. Put the learners in groups and ask them to make a list of what they see as a user for each event, and what the OS is actually doing behind the scenes. **(I)** Present the concept of a process and the different states a process can be in. Give learners an example of a process and ask them to identify the events that would move the process between different states. Get learners to act out the processes. Give each learners the name of a process. Students physically move between the states depending on a list of events that happen. Read the events one at a time and learners have to react and move depending on the event and if it impacts them. Demonstrate the use of virtual memory, paging and segmentation using diagrams on a computer where the data in memory can be split into pages, and then dynamically moved between the HDD and RAM as different instructions and data are required. Give learners a series of facts about virtual memory, paging and segmentation and ask them to identify which of the three concepts they relate to. **(F)**
16.2 Translation Software **(KC4)** **(KC1)** **(KC2)**	Explain how an interpreter can execute programs without producing a translated version. Describe the various stages in the compilation of a program.	Provide learners with a short program. As each stage of compilation is explained, ask learners to perform the actions on the program they have been given – this could be done in pairs. At the end of each stage, ask learners to compare their results. Give learners a BNF diagram and ask them what they think it means – discuss the answers and lead them towards its function and how it works. Present learners with a scenario (syntax description) and a BNF diagram that is incomplete – ask learners to complete the diagram. **(F)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Use Backus-Naur Form (BNF) and syntax diagrams to express the grammar of a language. Use Reverse Polish Notation (RPN) to carry out the evaluation of expressions.	Give learners a scenario and BNF diagram that includes errors and ask them to identify them and correct them.**(F)** Give learners program code and a BNF diagram and ask them whether the code is valid/invalid and to justify their answers. Present learners with a scenario and ask them to create a BNF diagram from it. **(I)** Give learners a BNF diagram and ask them to change it, to introduce more options, and/or restrict the grammar further. **(F) (I)** **Extension:** Ask learners to write a program to check the syntax of a program that is input against BNF statements. Discuss the use of brackets and priorities in mathematical calculations. Ask learners how computers 'know' which action to perform first. Show learners how RPN can tell computers the order to perform the calculations in. Show learners how stacks are used to represent the RPN expression and how these are evaluated. Give learners a blank 'stack' diagram and an RPN expression; ask them to put the items on the stack and then perform the calculations one step at a time – getting the item(s) from the stack, performing the calculation and putting the output back. Give learners a mathematical expression and ask them to put it into RPN. Repeat but reverse, giving learners an RPN expression and asking them to turn it into a normal mathematical expression. **Extension:** Ask learners to write a program that takes an expression and produces the RPN expression for it.
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q4		

Unit 17 Security

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
17.1 Encryption, Encryption Protocols and Digital certificates **(KC3)** **(KC1)**	Define the key terms associated with encryption. Describe the use of encryption, symmetric and asymmetric encryption. Explain the purpose and use of SSL and TLS. Explain how digital certificates are used.	Recap encryption from 6.1 Data security, i.e. purpose and simple encryption algorithms. With learners working in pairs, give one learner the role of sender and the other that of receiver. Get learners to act out symmetric and then asymmetric encryption on a method, sending the message between each other and encrypting and decrypting the message. Discuss the need for secure connections when online and how SSL and now TLS allow for secure transmissions between servers and browsers. Relate back to the use of symmetric encryption and the use of handshakes. Ask learners if they have ever had a website blocked because its digital certificate is invalid, or out of date. Discuss the contents of a digital certificate. Ask learners to find an example certificate online and describe the data items included. **(I)** Give learners facts about the different methods of encryption, the security protocols and digital certificates. Ask learners to identify which method they apply to. **(I)** Put learners into pairs. Give each pair a scenario where data needs to be transmitted securely. Ask them to identify an appropriate method of communication and which method they will use to ensure this is secure. Ask learners to explain their choice to the rest of the class and to justify their decision. **(F)** **(I)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q5		

Unit 18 Artificial Intelligence (AI)

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
18.1 Artificial Intelligence (AI) (graphs) **(KC1)** **(KC2)**	Explain how graphs can be used to aid Artificial Intelligence (AI). Use A* and Dijkstra's algorithms to perform searches on a graph.	Give learners a graph with labelled nodes. Ask learners to identify all the different ways they could get from one node to another. Ask learners what the graph and nodes look like (a map, state-transition diagram, etc.) – refer to abstraction from 9.1 Computational Thinking Skills and what the nodes could represent. Explain the difference between a directed graph and a non-directed one. Add weights to the graph and ask learners to find the most efficient path, and write down the list of instructions that they followed to find the shortest distance. Give learners a scenario where a graph could be used, and ask learners to create a graph for the scenario. Ask learners to compare their graphs and identify any differences. **(I)** Show learners how to perform an A* search and complete a suitable table to keep track of the nodes visited. Present learners with a graph with a partially completed A* search and ask learners to work together to complete it. Give learners a graph and ask them to perform an A* search. **(F) (I)** Show learners how to perform Dijkstra's algorithm to perform a search on a graph and complete a suitable table to keep track of the nodes visited. Give learners a graph with a partially completed Dijkstra's search and ask learners to work together to complete it. **(I)** Give learners a graph and ask them to perform Dijkstra's search. **(F)** Explain the use of heuristics in the searching algorithms and how they can help decrease the search time. Ask learners to perform an A* and Dijkstra's search on the same graph and keep track of the number of nodes visited, and the final solution found. **Extension:** Ask learners to find some further examples where heuristics are used, and if there are any searches that are more efficient than A* and Dijkstra. **(I)**
18.1 Artificial Intelligence (AI) (applications) **(KC2)** **(KC1)**	Explain how artificial neural networks help with machine learning.	Put learners in groups and give each group an AI technique and/or method of learning. Ask each group to research their technique and create a presentation for the rest of the class including case studies of where the techniques were used. **(I)** Give learners case studies of current developments in AI, applications of AI, etc. Ask learners to work in groups, each with a case study, and identify the AI techniques that may be used, and how the application was developed.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	Explain the use of Deep Learning, Machine Learning and Reinforcement Learning and the reasons for using these methods. Describe back propagation and regression methods in machine learning.	Use a practical example to show how computers can predict based on stored rules and past events. Link this to the different types of learning and show learners how the computers 'decide' which route to follow, and how these rules change. Link to relatable scenarios, such as board games or route finding apps. Simulate machine learning, for example with a game of noughts and crosses, or rock paper scissors. The computer's moves can be stored as instructions on paper, or using software, such as in a spreadsheet. The learners then play the game, using the stored instructions as moves and then change these each time a move is successful or otherwise. Students can then play their AI games against each other. **(I)** **Extension:** Ask learners to write a program to simulate one of the games discussed, i.e. to create the game of noughts and crosses with the computer learning its own moves. **(I)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q6		

Unit 19 Computational Thinking and Problem-Solving

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
19.1 Algorithms (searching and sorting) **(KC1)** **(KC2)** **(KC5)**	Describe a linear and binary search. Write algorithms to implement a binary and linear search. Describe an insertion sort and a bubble sort. Write algorithms to implement an insertion and bubble sort.	Recap linear search. Show learners how to perform a binary search. Provide learners with cards with numbers on, and tell them to put them face down in ascending order. Learners should follow the binary search instructions to find a specific card. Give learners program code for a linear and binary search and ask them to trace the algorithms. Provide learners with the searching algorithms with sections missing and ask learners to complete the missing statements. **(F)** Give learners the searching algorithm statements in individual lines, in a different order. Ask learners to put the algorithm into the correct order. **(F)** Present learners with the searching algorithms with errors in and ask learners to correct the errors. **(F)** Give learners inefficient searching algorithms (e.g. excess memory usage, iterations etc.) and ask learners to improve the efficiency of the algorithms. **(I)** Give learners a program to create which requires the searching algorithm(s) and ask learners to implement the program. **(F)** Recap bubble sort. Show learners how to perform an insertion sort. Provide learners with cards with numbers on, tell them to shuffle them and put them face down in a row. Students should then follow the instructions for an insertion sort on their cards, turning them over when required. Give learners program code for a bubble and insertion sort, learners to trace the algorithms. Present learners with the sorting algorithms with sections missing. Learners should read the algorithms and add the missing statements. **(F)** Give learners a program that needs a sorting algorithm and ask learners to implement it with both algorithms. **(F)** Provide learners with inefficient sorting algorithms and ask learners to make them more efficient. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
		Provide learners with sorting algorithms that include errors. Ask learners to find the errors and correct them. **(F)** Give learners a searching/sorting algorithm without a name and meaningless identifiers and ask them to identify it.
19.1 Algorithms (Abstract Data Types) **(KC1)** **(KC2)** **(KC5)**	Describe linked lists, stacks, queues and binary trees. Write algorithms to find items in a linked list and a binary tree. Write algorithms to insert items into a stack, a queue, a linked list and a binary tree. Write algorithms to delete an item from a stack, a queue and a linked list. Explain how an ADT can be implemented using a built-in data type and another ADT, and write algorithms to implement this.	Recap stacks, queues and linked lists theory. For each, ask learners to explain how to add and delete items. Ask learners to develop this explanation into structured English, and then into pseudocode. Give learners a partial program that defines the stack/queue/linked list and ask learners to write subroutines to add and delete an item from the stack/queue/linked list. **(F) (I)** Show learners how to create a binary tree. Draw a tree on a board and ask learners to add items one by one to the tree (e.g. by writing the node and branch). Define the key elements of a tree (e.g. root, branch, leaf). Ask learners to identify where these elements are on a tree. Ask learners to explain how to add an item to a binary tree. Students should then develop this into structured English, then pseudocode. Give learners a program with a linked list implement (e.g. as an array) and ask learners to implement the insert and delete subroutines. **(F) (I)** Show learners an example of a linked list and a binary tree. Ask them to work out how to find an item within it efficiently. Students can then develop this into structured English and then pseudocode before implementing their algorithm. **(F) (I)** Give learners example ADTs represented on paper and ask learners to add and remove data to/from the diagrams. Recap user-defined data types, and arrays and show learners how one ADT can be represented in each appropriate method (including other ADTs). Ask learners to implement this structure – it could be partially provided. Ask learners to take this knowledge and apply it to another ADT and work out in groups how to implement this ADT in other data types (and ADTs). **(I)** Give learners scenarios and ask them to work out which ADT is most appropriate. Ask learners to justify their choice. Students then need to write a program to implement the ADT for the scenario. **(F) (I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
19.1 Algorithms (performance) **(KC1)** **(KC2)** **(KC5)**	Explain the use of Big O notation to specify time and space complexity. Compare algorithms on criteria such a time taken and memory used.	Show learners two programs that perform the same actions but with different levels of complexity, e.g. time and/or space. Ask learners to explain what the differences are and how these could impact the program. Introduce time and space complexity and Big O notation, one complexity at a time. Relate this back to the standard algorithms covered, e.g. searching and sorting algorithms. **Extension:** Students can write the different algorithms that perform the same function and either get the compiler to calculate the execution time, or write a timer into the program. Students can then change the number of items, item being searched for, order of the items, etc., to see how it changes the time it takes the program to run. Give learners a list of algorithms and complexities. Ask learners to match the time and space complexities to the algorithms. **(F)** Give learners a list of different complexities and ask learners to put them into ascending/descending order of time for a given scenario (e.g. small number of items). **(F)** Put learners into groups and give each details of a scenario. Provide a list of possible algorithms along with the time and space complexities for each. Ask each group to choose the most appropriate algorithm and to justify it based on the complexities. **(F) (I)**
19.2 Recursion **(KC1)** **(KC2)** **(KC5)**	Identify the essential features of recursion. Write and trace recursive algorithms. Compare the use of recursion to iteration. Describe what a compiler has to do to translate recursive programming code.	Show learners a recursive algorithm. Ask them to work out in pairs what is different about this algorithm to other algorithms. Discuss the differences with a recursive algorithm. Ask learners to trace the algorithm and work out what happens when it is run. **(F) (I)** Give learners a series of recursive algorithms and ask them to identify the key features that each of the algorithms have – discuss the answers and reduce the list to the essential features of recursion. Discuss what might happen if one (or more) of these features is missing, e.g. it will never stop if there is no stopping condition. Show learners how to follow and then unwind a recursive algorithm, keeping track of the function/procedure call and return values. Give learners a range of recursive algorithms to follow. **(I) (F)** Give learners a recursive algorithm in program code. Ask them to write the same program using iteration in your chosen programming language. This could be completed in pairs to begin with. Repeat with a number of algorithms and ask learners what the common features/actions are that they follow when converting the algorithms – create a list of steps for learners to follow when converting an algorithm. Repeat this with the reverse, i.e. converting an iterative program to a recursive one. **(I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
		Discuss the benefits of iteration and recursion, both for the programmer and on the running of the program. Give learners a list of these benefits and drawbacks and ask learners to identify if they belong to iteration or recursion. **(F)** Discuss the use of a stack in the recursive calls when the compiler runs the code. Ask learners to implement this using a diagram of a stack and a recursive algorithm, putting the parameters, etc., onto the stack and then calling them back when it is unwound.
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q7 9618/4 Specimen paper Q1, 2, 3		

Unit 20 Further Programming

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
20.1 Programming Paradigms **(KC1)** **(KC2)** **(KC5)**	Explain what is meant by a programming paradigm. Write low-level code that uses various addressing modes. Write imperative programming code that uses constructs, procedures and functions.	Give learners example low-level programs to read and describe what each step does. They can also trace the algorithms using trace tables. **(F)** Identify key sequences that learners may have to write and ask them to read and write this code, for example incrementing a counter. Give learners a range of programs (in increasing complexity) written in assembly language that have lines of code (or sections) missing and ask learners to complete them. **(F)** Present learners with programs in assembly language that have errors and ask them to correct them. **(I)** Give learners a description of a program, or example pseudocode algorithms, and ask them to write these in assembly language. Some of these could be performed in pairs to begin with. **(F) (I)** Show learners a mixture of lines of assembly language code. ask them to select and order the ones necessary to solve a problem. **(F)** Give learners descriptions of programs to write using procedural program code, increasing in complexity. Students could also work in groups to produce a program by each developing a subroutine that they can then combine to create a complete project. **(I)** Provide learners with procedural program code that contains errors and ask learners to run the algorithms, find and replace the errors. **(I)**
20.1 Programming Paradigms (OOP) **(KC1)** **(KC2)** **(KC5)**	Use the terminology associated with OOP. Write program code to solve problems by designing appropriate classes	Introduce objects related to physical objects/people and how they have characteristics (attributes/properties) and can perform actions (methods). Give learners a real-life object/person and ask them to identify the attributes and methods. Give learners a scenario and ask them to work in pairs/groups to identify the objects, attributes and methods that may be required by the program. Students can then program the classes individually. **(F) (I)**

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
	and making use of OOP techniques.	Give learners example OOP code with errors in and/or missing elements; for example, a missing constructor. Ask learners to find and correct the errors and/or complete the missing items. **(F) (I)** Give learners a class diagram and ask them to create a program that meets the requirements of the class. **(F)** Discuss inheritance and relate it to real-life objects again, for example vehicles, food, animals, etc. Give learners example scenarios and ask them to work in pairs to identify the objects and inheritance. **(I)** Show and discuss with learners incomplete class diagrams. Ask learners to explain the inheritance between the classes. **(F)** Give learners the terms relating to OOP and definitions and ask them to match them. This could also be done as a crossword or other puzzle. **(F)** Give learners a program that has classes pre-defined and ask learners to implement a program that defines objects of the class and makes use of them. **(I)** Put learners into group and give them a project that requires an OOP. Students need to split the program development between themselves and decide on the classes, objects, etc., that they need. **(I)**
20.1 Programming Paradigms (Declarative) **(KC1)** **(KC2)** **(KC5)**	Read and write program code to solve problems by writing appropriate facts and rules.	Give learners a program written in a declarative language. Ask learners to trace the code and work out what each line means. Discuss their findings and how facts are declared in the language. Give learners a series of questions starting in English-style sentences, and then gradually introduce syntax for learners to read. Give learners an example program and ask them to add further facts to the program, and then run queries from it. **(I)** Give learners descriptions of rules and show them how to convert these into declarative language rules. Give learners some rules for them to explain and to check which facts meet them. Ask learners to change rules, correct errors in rules (particularly common errors such as ordering) and then write new rules that are given to them in sentences. **(I)** Give learners a scenario and ask them to design and create a declarative language for the scenario. **(F) (I)** **Extension:** Look at systems that make use of declarative languages and get learners to undertake case studies of these systems and why they use declarative languages.

Syllabus ref. and Key Concepts	Learning objectives	Suggested teaching activities
20.2 File Processing and Exception Handling **(KC1)** **(KC2)** **(KC5)**	Write code to perform file-processing operations. Explain the importance of exception handling. Write program code to use exception handling.	Ask learners to extend programs they have already created to store data e.g. objects created, in files, and to then reload these when the program starts again. **(I)** Give learners a game scenario and ask them to program the game along with a high score table that needs storing externally. **(I)** Learners can create a program where users need to create an account, which must be stored in a file, then they can only use the program if they give valid log on details. **(I)** Recap types of file access and ask learners to write programs that make use of these different methods. Show learners a program that produces an error and crashes. Discuss the problems with this format (i.e. it crashes and the program needs to be restarted). Ask learners how this could be handled to avoid the program crashing and introduce exception handling. Show learners how to catch exceptions in your chosen language. Provide them with written exceptions and ask them to read the code and explain each part of it. Give learners a program and ask them to add exception handling routines to it. **(F)**
Past and specimen papers		
Past/specimen papers and mark schemes are available to download at www.cambridgeinternational.org/support **(F)** 9618/3 Specimen paper Q8 9618/4 Specimen paper Q1, 2, 3,		

Cambridge Assessment International Education
The Triangle Building, Shaftesbury Road, Cambridge, CB2 8EA, United Kingdom
t: +44 1223 553554
e: info@cambridgeinternational.org www.cambridgeinternational.org