Teacher Guide

Cambridge IGCSE® and Cambridge O Level Computer Science

0478 and 2210

For examination from 2016

Cambridge Secondary 2

CAMBRIDGE International Examinations

Version 1
Updated: 09.03.16

Contents

Introduction

The purpose of this teacher guide

This teacher guide is designed to introduce you to the Cambridge IGCSE (0478) and Cambridge O Level (2210) Computer Science syllabuses and the related support materials available from Cambridge. It will help you to organise and plan your teaching. It also offers advice and guidance on teaching strategies, how to develop your learners' programming skills and how to prepare your learners for the final assessment.

What do I need to get started?

When planning your course, your starting point should be the **syllabus**. This contains information not only on the curriculum content but also the overall aims and assessment objectives. It gives details of the two papers, the grade descriptions and additional information. It is most important that you become thoroughly familiar with all parts of the syllabus document.

You will then need to devise a **scheme of work**. To do this, you need to think how you will organise the time that you have available to help learners to understand and learn all of the facts and concepts required by the syllabus, and to develop the necessary skills (such as programming). Cambridge provides a sample scheme of work that you could use as a starting point but you will undoubtedly want to produce your own at some point. (Extracts of the Cambridge published scheme of work are included in appendices B and C of this guide.)

Your scheme of work will help you to determine what **resources** you will require to deliver the course and this will help you to build up teaching, learning and reference resources such as text books, worksheets and sample programs.

You should make sure at an early stage that you have access to our secure online support for Cambridge teachers called **Teacher Support**, http://teachers.cie.org.uk. This provides a wide range of resources to help you, including past examination papers, mark schemes, examiner reports, example candidate responses, a resource list and community resources. All of these are invaluable in helping you and your learners to understand exactly what Cambridge expects of candidates in examinations, and will help you to prepare your learners appropriately.

Here is a checklist to help you get started.

Checklist

- Have you read the syllabus and checked that it is for the correct year?
- Have you looked at the Cambridge website and Teacher Support?
- What support materials are you going to use?
- What local resources are available to use?
- What school resources are available to use?

Section 1: Syllabus overview

1.1 Aims

The Cambridge IGCSE and Cambridge O Level Computer Science syllabus aims are to develop:

- computational thinking, that is to say thinking about what can be computed and how, and includes consideration of the data required
- understanding of the main principles of solving problems by using computers
- understanding that every computer system is made up of sub-systems, which in turn consist of further sub-systems
- understanding of the component parts of computer systems and how they interrelate, including software, data, hardware, communications and people
- skills necessary to apply understanding to solve computer-based problems using a high-level programming language.

Computer science is the study of the principles and practices of computation and computational thinking and their application in the design and development of computer systems. This syllabus aims to encourage candidates to develop computational thinking, that is thinking about what can be computed and how, and includes consideration of the data required. Learning computational thinking involves learning to program (to write computer code) which is the means by which computational thinking is expressed.

The assessment is by written papers, but the learning should be done in a mainly practical way: problem solving and programming. Questions will require the candidate to think, use knowledge with understanding and demonstrate understanding gained through practising practical skills. Questions will not revolve around pure recall.

1.2 Curriculum content

Sections	Topics
Section 1 **Theory of Computer Science**	1.1 Data representation 1.2 Communication and Internet technologies 1.3 Hardware and software 1.4 Security 1.5 Ethics
Section 2 **Practical Problem-solving and Programming**	2.1 Algorithm design and problem-solving 2.2 Programming 2.3 Databases

Section 6 of the syllabus lists the content of the curriculum. It is here that you will find details of exactly what your learners will need to know, to understand and be able to do when they sit the examination papers at the end of the course. The content is presented as a series of topics.

For section 1 each topic is divided into sub-topics that show what is to be taught, how it is to be taught and the computational uses required.

For section 2 each topic is divided into sub-topics that show what is to be taught and the practical skills that are to be developed.

1.3 Assessment objectives

The Assessment Objectives (which can be found in section 5.2 of the syllabus) are statements about what will actually be tested in the final examinations. Each question or task that is set in the examination relates to one or more of these Assessment Objectives (AOs).

There are three Assessment Objectives:

AO1 Knowledge with understanding

Candidates should be able to:

- recall, select and communicate knowledge and understanding of computer technology

Knowledge and understanding are clearly linked. Learners may, for instance, be able to recall the description of a computer virus as 'self-replicating code'. If, however, they do not understand what this means, they may not be able to answer questions which are based on the concept of a virus but do not use the exact words.

The second assessment objective includes application. It is stated in the syllabus as:

AO2 Application

Candidates should be able to:

- apply knowledge, understanding and skills to solve computing or programming problems

The ability to apply knowledge is a key skill in computer science. Learners need to be able to identify and solve problems in logical manner. They must be able to write and interpret algorithms using pseudocode, flowcharts and a high-level programming language.

In order to develop the skill of programming, learners need plenty of practice in writing programs in a high level programming language. Learners may use a high-level programming language of their choice; no particular programming language will be assumed in this syllabus. Centres may wish to decide the high-level programming language that will be used by all the learners. This could depend upon the software available and the expertise of the teachers.

AO3 Evaluation

Candidates should be able to:

analyse, evaluate, make reasoned judgements and present conclusions.

It takes time and a considerable amount of practice to develop the skills of analysis, evaluation and making reasoned judgements. You can help your learners build up these skills in a variety of ways. These include:

- asking learners to make short presentations in which they consider for example, whether to use a high-level programming language or a low-level programming language to provide a solution to a problem and make a recommendation having considered the requirements of the solution
- providing algorithms showing different solutions for the same problem and asking learners to discuss the effectiveness of the solutions
- setting past questions for learners to answer.

1.4 Assessment structure

Candidates sit two papers. Paper 1 tests the theory of computer science. The duration of the paper is 1 hour 45 minutes. It has a weighting of 60% of the total available marks.

Paper 2 tests problem-solving and programming, it consists of two sections. In section A, there is one question set on the pre-release material issued a few months before the examination. The duration of the paper is 1 hour and 45 minutes and has a weighting of 40% of the total available marks.

Components	Weighting
Paper 1 Theory 1 hour 45 minutes This written paper contains short-answer and structured questions. All questions are compulsory. No calculators are permitted in this paper. 75 marks Externally assessed.	60%
Paper 2 Problem-solving and Programming 1 hour 45 minutes This written paper contains short-answer and structured questions. All questions are compulsory. 20 of the marks for this paper are from questions set on the pre-release material. No calculators are permitted 50 marks Externally assessed.	40%

The testing of the Assessment Objectives is distributed across the two papers as shown in the table below. Both papers assess all three AOs.

Assessment objective	Paper 1	Paper 2	Weighting for qualification
AO1	32%	8%	40%
AO2	16%	24%	40%
AO3	12%	8%	20%
Total	60%	40%	100%

Section 2: Planning the course

This section first considers the benefits of planning; it then explores the process of planning on three levels, each of increasing detail. These include planning the overall course, planning the schemes of work (i.e. the teaching units) and planning the individual lessons. Examples of schemes of work, two lesson plans and a lesson plan template are provided in the appendices, to illustrate the principles explained in this guide.

2.1 Benefits of planning

Planning provides a number of significant benefits. These include:

- increasing the likelihood that all aspects of the syllabus will be covered
- helping to develop a logical structure to the course
- making you think about creating a variety of activities in lessons
- helping you build in formative and summative assessment. Formative assessment occurs throughout the course and influences subsequent teaching and learning. It involves gathering information on what learning is taking place through, for instance, marking class work and homework and feeding back to learners on their performance. Summative assessment establishes what progress a learner has achieved. It is often used to report to other institutions and to parents.

2.2 Long-term planning

The purpose of the long-term plan is to set out a framework that ensures the whole syllabus (including the development of the AO2 and AO3 skills) is covered within the time that you have available.

Each Centre will need to consider a number of factors in the light of its particular circumstances. These include:

- the amount of teaching time available for the whole duration of the course (IGCSE and O Level syllabuses are designed on the assumption that learners have about 130 guided learning hours* over the duration of a two year course)
- the number and length of lessons that you expect to have available (remember to take into account time lost to internal examinations or other activities that will take learners away from your lessons)
- the number of lessons available for practical work in a computing laboratory
- taking account of any prior knowledge learners may have
- whether you are sharing the teaching with a colleague or colleagues
- the homework policy of your educational establishment
- the assessment policy of your educational establishment, including when you can set mock examinations and when the Cambridge examinations will fall.

'Guided learning hours' refers to the time that the learner spends being directly taught by the teacher, or carrying out supervised work or directed study. In addition, learners will need to spend some time in private study.

All of these factors vary greatly between Centres. It is therefore most important that you develop your plans to suit your particular circumstances.

You will almost certainly find that you need to review your long-term plan each year. You will need to take account of any technical updates that have been published for the next year to take account of emerging technologies relevant to the computer science syllabus content. There may be changes in the hardware and software available in your school for learners. You may find that some topics took you longer than expected, while others were covered more quickly. You may decide to change the sequence in which you originally taught certain topics, perhaps because it became clear that learners needed to acquire more underlying knowledge and understanding before they were able to deal effectively with a particular set of learning objectives. However, the order of the syllabus in this particular subject does lend itself largely to a good order of teaching. Ideally, all teachers within the computer science department should be involved in reviewing how well the long-term plan is working and suggesting how it could be improved.

2.3 Medium-term planning (creating a scheme of work)

A scheme of work should indicate how you intend to cover all the learning outcomes. It should contain suggested teaching activities and related learning resources. It should also contain details of how you will help learners develop the AO2 and AO3 skills in the syllabus.

Key factors to consider when planning your scheme of work:

- the order of teaching. You do not have to follow the syllabus in order but many teachers do so as it has a logical structure, building skills and knowledge in a clear hierarchy and preparing learners for future learning
- the abilities of your learners. This will influence the pace at which you cover the course and the activities you use. If you have a mixed ability group, and most classes do include a range of abilities, you will need to develop some differentiated tasks (more on this in section 3.3 below)
- teaching style. You will have your own teaching style and your scheme of work should reflect this
- programming skills. You will be skilled in writing programs in certain high-level languages and your scheme of work should reflect this
- the resources available to you. There will be some resources you can access straight away and other resources you will build up over time
- assessment opportunities. You need to build in opportunities to assess the learners' progress at regular intervals and gauge their understanding of key concepts and common errors related to these
- opportunities for cross-curricular links
- practical programming and other computer based activities
- suggested homework and extension activities
- building in flexibility (more on this in section 2.6 below)
- working with other teachers to plan a scheme of work, collaborative documents can be easily prepared using the Internet

Cambridge provides a sample scheme of work on the secure online support facility for Cambridge teachers, Teacher Support, http://teachers.cie.org.uk. (You will need a password, obtainable from your Examinations Officer, to get access to the website.) Extracts from this published scheme of work are provided in Appendices B and C; it is important to understand that this scheme of work is intended only as an example, and you are not obliged to follow it. Each educational establishment will wish to develop their own scheme of work, to suit their particular circumstances and their own learners. It is always good practice to involve everyone in the department in the construction of the scheme of work.

2.4 Short-term planning (creating lesson plans)

A short-term plan is an outline of what you intend to do in a particular lesson, or perhaps a small group of lessons. You may decide to plan lessons on a weekly basis. In practice, it is unlikely that you will have the time to plan all lessons in great detail. It is, nevertheless, worth planning key lessons in depth. Such lessons may include those which start a new topic, those which introduce a new skill and any areas in which you are going to be observed.

Key factors to consider when planning lessons:

- what you want the learners to be able to do by the end of the lesson – what learning you want to have taken place and what skills you want learners to have developed (AO2 and AO3)
- the benefits of recapping the learning achieved in the previous lesson and outlining the learning objectives
- how you are intending to help learners to achieve these goals
- how you will start and end the lesson
- what activities are to be used. Variety can create interest, keep learners alert, draw on and develop a range of skills
- how you will ensure that all learners, no matter what their ability, will be suitably stretched and occupied throughout the lesson
- what resources you will need (e.g. worksheets, computing resources, video clips, etc.)
- the approximate timings you expect each stage of the lesson to take
- how to assess what learning has taken place
- the benefits of recapping the key points at the end of the lesson.

You can find sample lesson plans in appendices D and E of this guide. Most of the categories in the template provided are self-explanatory but you may want to clarify the difference between the following items:

Teaching aims

Teaching aims are the general aims you set for yourself to achieve during the lesson.

Lesson objectives

Lesson objectives are what you are aiming for the learners to be able to do by the end of the lesson. You may wish to share the lesson objectives with the learners by writing them on the board at the start of the lesson. You might also ask them at the end of the lesson whether they think the objectives have been achieved.

Syllabus assessment objectives

These are based on a combination of the assessment objectives and the curriculum content from the syllabus.

The aims and objectives are clearly connected. They are designed to set targets for you and the learners and to link these to the syllabus.

2.5 Reflection and evaluation of learning

A lesson plan should provide the opportunity for you to review how the lesson went. There are a number of questions you may wish to consider:

- What lesson activities went well?
- Why did they go well?
- Were all learners fully involved?
- What did not go well?
- What were the reasons why they did not go well?
- Were all the aims and objectives achieved?

You may also want to ask another teacher, on occasions, to observe one of your lessons to receive feedback and you could give out a learner questionnaire asking, for instance, which activities they have enjoyed and whether they are finding the feedback on their class work and homework useful.

2.6 Flexibility

A plan is a useful guide but it must not restrict what you do. You must be prepared to adapt the plan in the event of any changes in the influencing factors, availability of computing equipment and the rate of progress of learners. For instance, new learners may join part way through the course or teaching hours may be lost due to illness or bad weather.

Changes in the technology available for learners to use may provide an opportunity to explore particular topics. For example, if there are new computer screens being installed in your educational establishment, you may want to devote some lesson time to exploring the principles of operation and the benefits and limitations of the new screens when they have just arrived. You may also want to change the order of the coverage of the course if materials become available for use. For example, if old or non-working equipment is to be disposed of, learners can use this to investigate how a computer works on the inside.

How your learners are progressing is probably the main reason why you may need to adapt your planning. If the learners are finding some topics easier than expected whilst others more difficult, you may need to devote less time to the former and more time to the latter. You may also discover that certain activities work well with the learners whilst they are less responsive to other activities and this response may change from year to year.

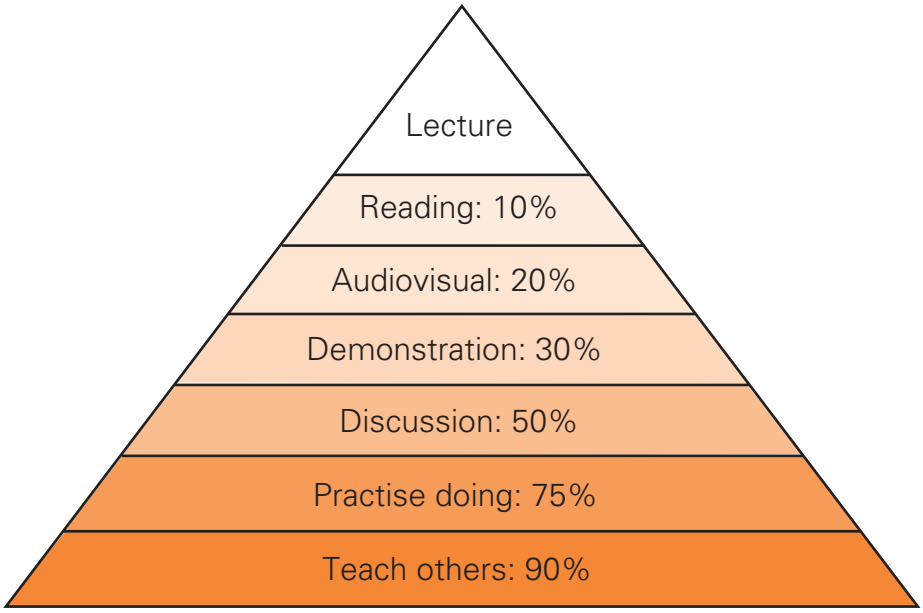
Section 3: Classroom practice

3.1 Active learning

Active learning is about learners being engaged in their own learning rather than simply being the passive recipients of knowledge as supplied by you. They are involved in a variety of activities that involve thinking, doing and talking, and develop their understanding of a topic by placing it in a variety of contexts.

Cambridge syllabuses aim to produce learners who are actively involved in their own learning. They should become self-confident computer scientists, able to identify and design solutions to problems. They should take an informed interest in the wide range of important computer-related issues in today's world.

Active learning will help to achieve these aims. Active learning techniques also increase learners' enjoyment of learning. Research shows that active learning is associated with much higher retention rates. Typical learner retention rates for different types of learning activities are illustrated in the 'learning pyramid'.



Learners should always have to process information. This can be achieved in a relatively straightforward way.

For example:

- you could ask learners to research sensors by referring them to a textbook or website
- you could write up the different types of sensors they have found on the board
- and then ask them, in groups, to consider what you could use a particular type of sensor for and why it would be appropriate.

Some teachers worry that giving learners more responsibility for their own learning will take too long. However, it can actually save time and more crucially, it can prove to be a better use of time.

There are a number of ways you can promote active learning. These include:

- **question and answer sessions.** This is a quick way of assessing learner understanding
- **group discussions.** These are more productive if you ask learners to research a topic beforehand. On some occasions, you can tell different learners to research different parts of the same topic for example some to research inkjet printers and others to research laser printers, whilst on other occasions, you could have open discussions
- **presentations.** These are particularly useful later in the course. A group of two or three learners can be instructed to research a topic for example copyright issues. Having given the presentation, the group could be required to answer questions from the class
- **wall displays.** Learners will learn as they are producing wall displays, and their presence in a classroom can make it attractive and can reinforce learning
- **worksheets.** These are a traditional way of getting learners involved in their learning. You can build these up over time and you might decide to produce some extension questions. As the course progresses, these worksheets should become more challenging. As well as reinforcing learning, worksheets should enable you to assess learning and help students prepare for the examination papers
- **role playing.** This type of activity might be carried out two or three times later in the course. There are a number of topics which might be explored using this approach. For example, 'the fetch-execute cycle' can be demonstrated by giving different learners the roles of the registers and addresses used, and by working through what happens to a short program. Learners could either be given briefing sheets on their roles or asked to produce them. This could follow the use of Little Man Computer (LMC) download: http://gcsecomputing.org.uk/lmc/lmc.html
- **structured examination type questions.** These can be used in a variety of ways. At the start of the course, you can get learners to work on these in pairs or groups.

Cambridge offers online tutor-led courses in this and other subjects where you can share teaching strategies and discover ideas for active learning in discussion with other teachers and the course tutor. It is a good place to tackle any difficulties you may have delivering a particular area or topic of the syllabus. Check the Cambridge website events pages to find out when courses are available and to register for them www.cie.org.uk/events

3.2 Practical learning

Programming is a skill that needs to be learnt and then practised regularly. Learners need at least weekly access to a computer in order to develop and practice programming skills. Whilst working in pairs may help to build confidence for the first few lessons, learners should aim to be able to confidently write, develop and test a program by themselves before the end of the course.

The choice of high-level programming language will depend upon the skill base of the teacher and the resources available. There are many suitable high-level programming languages that can be used, such as Visual Basic, Pascal/Delphi or Python. If your learners have not attempted programming before they start on the course then the concept of program writing could be introduced by using Scratch to code simple programs. There are many free resources available for teaching programming, including software and tutorials to download.

Before using any resources, free or purchased, with your learners; you should test them out on the hardware that is going to be used by your learners, to ensure that they work as expected and there are no problems. Using untried software with learners can bring up many different problems that may prevent learning taking place.

Tutorials also need testing and probably editing before use by your learners to ensure that:

- the instructions to be followed work
- the English used is suitable for learners whose first language is not English
- the tutorials cover what is required by the syllabus
- programs are tested with suitable test data

There are other computer science skills for your learners to practice during this course. Opportunities should be provided on a regular basis to recognise what skills should be applied to a problem and how to use these skills, for example:

- conversion between binary, denary and hexadecimal
- creation of logic circuits and production of truth tables
- devising algorithms to solve problems
- setting out these algorithms in standard ways using flowcharts and pseudocode
- systematically testing algorithms, showing the results in trace tables
- devising test data for testing

3.3 Adapting to different learning styles

All groups, to a certain extent, are of mixed ability and the ability of learners obviously changes over time. By encouraging active learning, it is possible to set work which will challenge the most able and will bring on the skills of those who might, initially, be struggling with the subject. Group work can be particularly useful. More able learners can benefit from being in a group which includes less able learners as they will learn by explaining points to their colleagues. Less able learners will benefit from having points explained by someone of a similar age.

The technique for dealing with mixed abilities is known as **'Differentiation'**. This method enables you to ensure that every individual learner in your class, no matter what their ability, is involved in tasks that are suitable for them, and that will enable them to make good progress. This is particularly important if you have a wide ability range in your class. You will need to think about how you can make sure that your brightest learners are being fully stimulated and stretched, while the least able can still feel fully engaged with the lesson, and make confident progress.

Differentiated work can be achieved in a variety of ways. Let's see how the technique might be applied to the ideas listed for active learning above:

- **question and answer sessions.** Target some questions to particular learners. You can mix these with questions thrown open to all those in the class
- **group activities.** You might start by dividing the class into a number of smaller groups and requesting that they try to complete a task. For example, they could find errors in a program or flowchart. You could listen to each group and ask each one to report to the whole class. If some learners are shy of speaking in public or less confident with the topic, you might decide to appoint a spokesperson for each group and give the role to the more confident learner. Over time you could share the role between two members in the group
- **presentations.** Different roles could be assigned. One learner might have the main responsibility for researching the topic, one for drawing up the presentation and one or two for giving the presentation. The quality of the presentation will be influenced by how well the learners work together. Differentiation might also be achieved by giving different groups, different topics

- **wall displays.** Again, you could assign different roles to learners. This activity can enable learners who have a good visual sense to do well, but they will need the help of those who have a good understanding of computer science
- **worksheets.** You can produce worksheets (for example on designing logic circuits, writing algorithms and writing programs) aimed at different abilities, or ones which include extension questions.
- **brainstorming.** You can target questions. You might also ask particular learners to lead the brainstorming, others to write up the ideas and others to produce mind maps based on the ideas.

Section 4: Preparing learners for final assessment

4.1 Use of past papers, mark schemes and principal examiner reports

It is important that learners should be familiar with the format and requirements of the two examination papers. They should not come as a surprise, as lack of familiarity can result in learners making mistakes. For instance, learners would disadvantage themselves by not completing the pre-release programming tasks before sitting the Paper 2 examination.

4.1.1 Past papers

These directly illustrate the requirements of the exam. Learners get to know the format of the exam, the type of questions asked and the style and type of command words used in different questions.

There are many ways in which past papers can help revision.

- **For practice:** This is useful to help learners assess whether they know and understand the subject matter or alternatively to identify gaps in their knowledge. Setting a whole paper is good practice for the examinations as learners also gain experience of working in exam conditions and within time constraints. You do not always have to use the whole paper. For Paper 2, for example, you could set either section A or section B and reduce the time accordingly to 45 or 60 minutes.
- **As a focus for revision:** You can see what level of detail is expected and this will help learners work out how much they need to know about different topics.
- **Understanding what the examiner will be looking for:** Ask learners to try marking someone else's response to a question to understand how an examiner will look at the answer. This will also enable them to see how others approach the same question.

A good number of past papers will be available over time on Teacher Support http://teachers.cie.org.uk as there are several variants of Papers 1 and 2 in each exam series.

4.1.2 Mark schemes

Probably the best way to use mark schemes is to set past questions (either single questions or whole papers) as a test for learners and check how well they have done by referring to the mark scheme. You might, sometimes, give them to learners to mark their own work or even each other's. You could also give them one past structured question part and the mark scheme on that question part and then ask them to write a similar question part and a corresponding mark scheme.

4.1.3 Principal examiner reports

These reports, found on Teacher Support http://teachers.cie.org.uk contain information on how candidates have performed in both examination papers. They give guidance on what examiners were looking for in each question, as well as which questions candidates have performed well on and any common mistakes and points of confusion. The reports are useful for both teaching and revision as you can pick up tips to help improve learners' understanding and exam performance.

4.2 Paper 1 – theory of computer science

This is a compulsory question paper, consisting of short-answer and structured questions set on Section 1 of the syllabus content. All questions are compulsory. Learners will answer on the question paper.

Most questions will not revolve around pure recall. Learners will be expected to apply their knowledge to a real situation, for example using binary digits for a digital alarm clock display.

4.3 Paper 2 – problem solving and programming

This is a compulsory question paper, consisting of short-answer and structured questions set on Section 2 of the syllabus content. All questions are compulsory. Candidates will answer on the question paper. 20 of the marks in this paper are from questions set on tasks provided in the Paper 2 Problem-solving and Programming pre-release material.

Teachers need to be aware that in order to prepare their candidates for this paper, they should plan for sufficient practical sessions within their lesson timetable and teach the contents of the section in a largely practical way. Learners will be expected to be able to program in a high-level programming language to be chosen by the Centre. The programming language should be procedural.

There will be some examining of knowledge with understanding, but most of the credit will be for using techniques and skills to solve problems. The examination questions will require candidates to have practical programming experience, including writing their own programs, executing (running), testing and debugging them. Knowledge of programming language syntax will not be examined; in all cases the logic will be more important than the syntax.

The Paper 2 Problem-solving and Programming pre-release material will be made available the January before the June examination, and the July before the November examination. It will also be reproduced in the question paper. Learners are not permitted to bring any prepared material into the examination.

Teachers are advised to encourage their learners to develop solutions to tasks using a high-level programming language(such as Visual Basic, Pascal/Delphi or Python). The purpose of the pre-release material tasks is to direct learners to some of the topics which will be examined in Paper 2. Teachers are expected to incorporate these tasks into their lessons and develop the appropriate skills. 20 of the marks in this paper are generated by questions testing these skills.

4.4 Command words used in examination questions

Most 'questions' do not end with a question mark but instead use a word that tells the learner what they need to do. For example, 'Explain', Describe', 'State', 'Give' and 'Give two differences between'. These words, often called **command words**, need to be read carefully by the learner, who needs to become familiar with exactly what they mean.

Many command words or phrases are self-explanatory for example:

- draw a line
- convert
- calculate
- show your working
- write an algorithm
- locate errors and suggest corrections
- complete a trace table/query-by-example grid

Other command words may need to be explained to learners. You will need to provide learners with questions, throughout the course, that use these command words appropriately and give them feedback on their answers that will help them gradually to learn the meaning of each one.

Name	this usually requires a technical term or its equivalent. For example 'name this type of signal.'
Describe	Means no more than it says: 'Give a description of...'. So, 'Describe a type of data entry error' requires a description of an error for example 'a transcription error occurs when character(s) are mistyped by a person entering data'.
Explain	This creates major difficulties for many candidates. A reason or interpretation must be given, not a description. The term 'Describe' answers the question 'What?' the term 'Explain' answers the question 'Why?'.
Using examples	Answers to questions involving this type of phrase must follow the instructions. 'Describe, with an example, a type of data entry error.' requires a description of an error with an accompanying example; for example 'a transcription error occurs when character(s) are mistyped by a person entering data e.g. a date of birth is entered as 06/06/9002 instead of 06/06/2009'.
State/Give	'State' or 'give' fall short of describing and amount to no more than making bullet points e.g. 'State **two** differences between Flash memories and CD-RWs.' ...'Flash memory is solid state, CD/RW is optical.'
Benefit or drawback	A benefit is simply the good outcome of an action or incident. A drawback is a bad outcome of an action or an incident.
Advantage or disadvantage	When asked to give an advantage or disadvantage, you will need to go a little further than just stating a good/bad outcome and offer some explanation as to why, providing an opposite that you can compare it to. For example, in answer to the question 'Give one advantage of writing code in high-level language', you might answer that 'It is easier for a programmer to understand than code written in a low-level language'. This helps define the advantage more clearly and gives it a context. Of course the same principle will apply if you are stating a disadvantage.
What your learners Can do in the examination	1. Read the question. 2. Understand the type of instruction you are being given. 3. If the question makes use of a specific scenario then make sure that all your answers are relevant to that scenario. 4. Decide on the information required but remember that many answers will require more than just a single word or a short phrase. If you have finished your examination well before the time allotted, you may have fallen into this trap. 5. Always use correct technical terms and avoid the use of brand names. Write about using a database management system to solve a problem rather than using 'Access'. 6. Decide how much information is required. – use the instructions for example 'declare **two** variables.' – look at the marks awarded. For example, 'Describe how the sensors and the microprocessor are used to maintain the correct conditions in the fish tank.' [4], will require 4 points.

Section 5: Resources

5.1 Teacher support

- Access to past papers, mark schemes and examiner reports.
- The current syllabus and syllabus updates.
- A searchable resource list.
- Schemes of work for each of the units.
- Answers to frequently asked questions.
- A discussion forum, moderated by a senior examiner.
- Application support booklets.
- Community resources which include resources teachers are prepared to share.
- Details about upcoming events and training sessions. Visit Teacher Support at http://teachers.cie.org.uk

5.2 Finding resources

Many resources can be found on Teacher Support. This includes a searchable resource list of published text books, syllabus documents and specimen papers, a scheme of work and links to websites.

5.2.1 Endorsed and recommended textbooks

Endorsed and recommended textbooks are available in our resource lists. When a title has been endorsed it means that it has been written to closely follow the qualification it related to, and is therefore suitable to be used as teaching material for those specific subjects. Recommended titles are useful as a reference resource when teaching or studying the subject but which have not been written specifically for the qualification they are linked to.

At the time of writing the following endorsed titles are being rewritten for the IGCSE/OL Computer Science Syllabus and are due to be published autumn 2014:

Cambridge IGCSE Computer Science Revision Guide, CUP India, (due 2015), D. Watson and H Williams

Cambridge IGCSE Computer Science, CUP, 2014, R Morgan, D. Scott and S. Lawrey forthcoming 2015

Cambridge IGCSE Computer Science, Hodder Education, 2015, D. Watson and H Williams

For an up-to-date list of endorsed and recommended titles please see Teacher Support.

5.2.2 Creating and sharing resources

Teachers within a department will find it useful to store resources centrally, perhaps on a shared folder on your intranet. This might contain:

- multiple-choice questions on particular topics
- sets of questions on particular topics
- sets of programming exercises at differing levels

- quizzes
- computer science crosswords
- video resources (e.g. items from BBC Click, clips from YouTube)
- filmed presentations given by learners.

Items can be added to (and deleted from) the shared departmental folder over time.

You can also share resources with other teachers. One way you can do this is through the community resources on Teacher Support.

5.2.3 Adapting resources

Past examination papers can be found on Teacher Support. You can use whole papers or parts of papers for your course.

Adapting resources is important in computer science as the subject matter is constantly changing. This is particularly important in terms of input devices, output devices, media and applications. You could provide your learners with current video clips, short programs and ask them to design questions. You could, for instance, tell them that the first question has to test knowledge, the second understanding and the third evaluation. This will help reinforce the skills they need to develop and the meaning of command words.

5.3 Training and professional development for teachers

5.3.1 Online training

Online training is occasionally available to Cambridge schools usually on a rolling schedule. Check the Cambridge website events tab to see when courses are running and to register.

Online tutor-led

Where available, these courses are led by a Cambridge expert. They focus on classroom practice. Teachers follow a three-unit programme over six weeks and can interact and share resources with teachers from other schools. Teachers on these courses often build up lasting links with their fellow teachers.

Online webinars

Where available these seminars are led over a short period by an expert and focus on specific issues such as syllabus changes or the recent examination session.

5.3.2 Face-to-face training

We run an extensive programme of short professional development courses across the world for teachers at Cambridge schools. Most workshops are run by a Cambridge trainer. These courses offer teachers the chance to update their knowledge, learn new skills and network with other teachers. Please check the Cambridge website events tab to see when courses are running and to register: www.cie.org.uk/events

5.3.3 Professional development for teachers

We also offer professional qualifications for teachers who want to develop their thinking and practice.

Learn more about the Cambridge International Certificate for Teachers and Trainers and the Cambridge International Diploma for Teachers and Trainers at: www.cie.org.uk/qualifications/teacher

Appendices

Appendix A: Sample long-term plan

Appendix B: Sample scheme of work for Unit 2: Numbers, processors and operating systems

Appendix C: Sample scheme of work for Unit 8: Programming concepts

Appendix D: Sample lesson plan 1: computer architecture and the fetch-execute cycle

Appendix E: Sample lesson plan 2: programming concepts

Appendix F: Sample lesson plan template

Appendix A: Sample long-term plan

This plan is based on a two year IGCSE/OL course, with the two examination papers being taken in the sixth and final term. It is a good idea to teach units addressing Paper1 and Paper2 in parallel, to balance theory with practical activity.

The units within the scheme of work (with suggestions for time allocations, based on a total allocation of about 130 hours) are:

Unit 1: **Introduction to computer systems** (15 hours)

Unit 2: **Numbers, processors and operating systems** (10 hours)

Unit 3: **Data communications and networking** (12–15 hours)

Unit 4: **Data integrity and security** (10 hours)

Unit 5: **Binary logic** (15 hours)

Unit 6: **Practical problem solving – structure diagrams, algorithms and flowcharts** (12 hours)

Unit 7: **Practical problem solving – pseudocode** (12 hours)

Unit 8: **Programming concepts** (12 hours)

Unit 9: **Databases** (9 hours)

Unit 10: **Use of pre-release material** (20–25 hours)

Term 1

Paper 1	Paper 2
Introduction to computer systems Focus on developing skills of knowledge with understanding	Introduction to Practical problem solving Introduction to programming

Term 2

Paper 1	Paper 2
Numbers, processors and operating systems Focus on developing skills of knowledge with understanding	Introduction to Practical problem solving, algorithms Programming concepts

Term 3

Paper 1	Paper 2
Data communications and networking Test Department review of learners' progress	Introduction to Practical problem solving, flowcharts and pseudocode Programming concepts Test Department review of learners' progress

Term 4

Paper 1	Paper 2
Review of previous year's work Data integrity and security Department to identify learners who need extra support	Review of previous year's work Databases Department to identify learners who need extra support

Term 5

Paper 1	Paper 2
Binary Logic Focus on developing examination techniques Learners to work through past papers and to draw up examination questions	Use of pre-release material Focus on developing examination techniques Learners to work through past papers to draw up examination questions

Term 6

Paper 1	Paper 2
Revision Practice exam papers Department review scheme of work for next year	Use of pre-release material Revision Practice exam papers Department review scheme of work for next year

Appendix B: Sample scheme of work Unit 2

Numbers, processors and operating systems

Recommended prior knowledge

In order to understand the role of an operating system, learners should have had practical experience of using at least one operating system with a Graphical User Interface (GUI). It is recommended that learners should have studied Unit 1 before starting this unit.

Context

This unit looks at the way in which numbers are represented within a computer system, the structure of the central processing unit and its functions, and the role of the operating system in managing the components of a computer system and interactions with the user.

Outline

This unit starts with binary and hexadecimal representation of numbers, leading to the von Neumann model of a computer system and the concept of a computer. This is illustrated practically by learner use of the Little Man Computer (LMC). The role of operating systems is then considered, including control of peripherals and the user interface. Learners will not be expected to know detail of any specific operating system.

Suggested teaching time

Based on a total time allocation of 130 contact hours for this Cambridge IGCSE/O Level Computer Science course, it is recommended that this unit should take about 10 hours.

Syllabus ref	Learning objectives	Suggested teaching activities	Learning resources
1.1.1	• understand binary notation and to convert denary numbers to and from binary	Teacher presentation to introduce the concepts of binary notation, e.g. using an automatic binary counter; binary number cards available (with worksheets etc.). **(W)** Learners convert denary numbers into binary and binary numbers into denary **(I)**; reinforce with a game such as the Cisco binary game. **(G)** This provides formative assessment of understanding. Learners answer previous exam/textbook questions on binary representation. **(I)**	Binary counter – for example: **www.mathsisfun.com/binary-decimal-hexadecimal-converter.html** Binary numbers at Computer Studies Unplugged: **http://csunplugged.org/binary-numbers** Cisco binary game: **http://forums.cisco.com/CertCom/game/binary_game_page.htm** *Cambridge IGCSE Computer Studies Revision Book* Chp 11.4 – sample questions in 11.6
1.1.1	• recognise the use of binary numbers in computer systems	Teacher presents the concept of the byte; class discussion about how the byte is used to measure memory size by introducing the concept of kb, Mb, Gb, Tb. **(W)** Class brainstorm to reflect on capacity of commonly found elements of computer systems such as hard disk drives, RAM, DVD, USB flash drives etc. (refer back to Unit 1 – types of memory). **(G)**	Simple comparisons at **www.bbc.co.uk/schools/gcsebitesize/ict/hardware/1datastoragerev2.shtml** Useful reinforcement material: **http://computer.howstuffworks.com/bytes.htm**
1.1.2	• understand hexadecimal notation and to convert hexadecimal integers to and from binary and denary • understand the significance of hexadecimal in computer systems	Teacher presentation on hexadecimal notation and its relationship to binary notation. Demonstration of the conversion of binary and denary to hexadecimal. **(W)** Learners convert positive hexadecimal integers to and from binary and to and from denary. **(I)** Class brainstorm to show understanding of the reasons for choosing hexadecimal to represent numbers, e.g. those stored in registers and main memory. **(W)** Learners answer previous exam/textbook questions on hexadecimal representation. **(I)**	Hexadecimal counter – for example: **www.mathsisfun.com/binary-decimal-hexadecimal-converter.html** *Cambridge IGCSE Computer Studies Revision Book* Chp 11.5 – sample questions in 11.6

Syllabus ref	Learning objectives	Suggested teaching activities	Learning resources
1.3.2 1.1.1 1.1.2	• show understanding of the basic Von Neumann model for a computer system and the stored program concept • describe the stages of the fetch-execute cycle • recognise the use of binary in computer registers • identify current uses of hexadecimal e.g. assembly languages and machine code, debugging etc.	Teacher presents basic concepts of computer architecture, including registers, and the fetch-execute cycle followed by demonstration via projector of the Little Man Computer (LMC). **(W)** Learners carry out simple low level tasks using LMC software – paired work is probably most effective. **(P)** Differentiation can be achieved by giving able learners more challenging tasks (examples available in quoted resources and by searching for the LMC tasks using e.g. Google).	Notes/presentation on computer architecture: **http://web.eecs.utk.edu/research/cs100modules/module1/index.html** Notes and animations of fetch-execute cycle: **www.eastaughs.fsnet.co.uk/cpu/execution-cycle.htm** Little Man Computer download: **http://gcsecomputing.org.uk/lmc/lmc.html**
1.3.6	• describe the purpose of an operating system • show understanding of the need for interrupts	Teacher presentation to include: • the idea of system software as different from applications software • general tasks and facilities of an operating system – for processor management, it is helpful to demonstrate Windows Task Manager • the role of the operating system (OS) in file management • how peripheral devices, such as keyboards and printers, must be controlled and responded to by the operating system • how communication between the computer and peripherals must be controlled and errors detected. **(W)** Learners (paired/grouped) to research:	*Cambridge IGCSE Computer Studies Coursebook* pp. 92–4 Cambridge IGCSE Computer Studies Revision Book 13 Several pages describing operating systems and their functions: **www.howstuffworks.com/operating-system1.htm** Windows, Linux, Android could be used as examples

Syllabus ref	Learning objectives	Suggested teaching activities	Learning resources
		• buffer • polling • interrupts • handshaking • checksum. Learners use their findings to create a short role play activity that demonstrates how each of these works **(G)**. Learners need to make their own notes on each of these after they have been acted out. **(I)** Class brainstorm to review learners' previous experience of operating systems with graphical user interfaces (GUI), and introduce the idea of a command line interface. **(W)** Discuss the main differences between command line interfaces and GUIs and their respective advantages and disadvantages. **(G/P)** Pairs of learners devise their own quiz questions (and answers) on this unit **(P)**; teacher selects one or two quizzes to test understanding of operating systems and their function. **(W)**	*Cambridge IGCSE Computer Studies Coursebook* pp. 102–5 Theory notes and activities on buffers (and drivers): **www.teach-ict.com/gcse_new/computer%20systems/buffers_drivers/home_buffers.htm** Old but still relevant article that compares interrupts with polling: **www.atarimagazines.com/compute/issue149/60_Interrupts_made_easy.php** *Cambridge IGCSE Computer Studies Coursebook* pp. 98–100 Notes on user interfaces: **www.igcseict.info/theory/1/uis/index.html** Quizzes to test understanding at: **www.teach-ict.com/gcse_computing/gcse_computing_quizzes.htm**

Appendix C: Sample scheme of work 2, Unit 8

Unit 8: Programming concepts

Recommended prior knowledge

Learners need to have studied Units 6 and 7 before starting this unit.

Context

This unit completes the process of converting an algorithm from an abstract idea to a working computer program. A range of different types of programming languages exist; this unit looks at the different levels of language and the processes for translation into machine code. It also provides learners with opportunities to convert algorithms into functional programs. Candidates are not expected to have expertise in any specific computer language but to understand the basic principles of syntax. Examination questions that require learners to write a program will include the syntax that needs to be used.

It is recommended that learners have the opportunity to write programs using two or three different programming languages. Examples might include Visual Basic, Delphi/Pascal, Python, Scratch and a control programming language. References to some of these are given in the resource lists below.

Outline

Following consideration of the concepts of sequence, selection and repetition, writing an algorithm as a flowchart and in pseudocode, and identifying and correcting errors in pseudocode, this unit looks at the need for high-level and low-level languages. It considers the use of assemblers, interpreters and compilers for translation of the code written by a programmer into machine code that can be used by the processor.

Learners have the opportunity of using a number of different high-level languages to produce working programs, to extend their knowledge of iteration by the use of FOR…NEXT, REPEAT…UNTIL and WHILE… DO loops and to incorporate the use of arrays into their programming.

Teaching time

Based on a total time allocation of 130 contact hours for this IGCSE course, it is recommended that this unit should take about 15 hours.

Syllabus ref	Learning objectives	Suggested teaching activities	Learning resources
1.3.7	• show understanding of the need for both high-level and low-level languages • show understanding of the need for assemblers when translating programs written in assembly language	Brainstorm the nature of a program and its requirements (data input and output; manipulation of data of various types and structures; sequence, selection, repetition and subprogram intercommunication; the concepts of totals and counting). **(W)** Teacher introduces learners to different types of programming languages by considering: • historical origins of computer programming in machine-specific types of language (machine language and assembly language) • the characteristics of these languages • the need for an assembler translation program for assembly language • why they are still used for certain applications. **(W)**	*Cambridge IGCSE Computer Studies Coursebook* pp. 255–9 *Cambridge IGCSE Computer Studies Revision Book* Chp 8.1
	• show understanding of the need for compilers when translating programs written in a high-level language • show understanding of the use of interpreters with high-level language programs	Learners research the characteristics of high-level languages; the need for compiler and/or interpreter translation programs for these languages; why they are preferred for many applications. **(G)/(I)**	*Cambridge IGCSE Computer Studies Coursebook* pp. 257–9

Syllabus ref	Learning objectives	Suggested teaching activities	Learning resources
			Extension work: • History of compiler writing: **http://en.wikipedia.org/wiki/History_of_ compiler_writing** • First high-level language to have a complete compiler: • **http://en.wikipedia.org/wiki/Fortran** • The first programming language to express operations using English-like statements: • **http://en.wikipedia.org/wiki/FLOW-MATIC**
2.2.1	• understand and use the concepts of sequence, selection, repetition, totalling and counting • use predefined procedures/ functions	Learners investigate programming concepts using a number of different easy-to-use high-level computer programming languages Introduction to programming with Scratch – teacher presentation to cover: • different data types and their declaration; • iteration, counting and totalling – implementation of some examples previously devised as pseudocode representations; • calling procedures/functions/sub-routines. **(W)** Followed by a range of learner practical activities **(G)/(I)**. These can be differentiated by task to provide appropriate challenge for learners. Repetition of the previous sequence of activities using • a control programming language (e.g. GO, Logo, Flowol) • a more conventional procedural language such as V-Basic, Python, Pascal etc.	Scratch – a simple programming language that makes it easy to create animations, games, music, interactive stories, etc. without the need to learn complex syntax: **http://scratch.mit.edu/** Some simple tasks in Scratch: **www.teach-ict.com/programming/scratch/scratch_home.htm** Flowol website: **www.flowol.com/Default.aspx** The Python website: **www.python.org/** Some LOGO websites and ideas: **www.mathcats.com/gallery/logodownloadinfo.html**

Syllabus ref	Learning objectives	Suggested teaching activities	Learning resources
2.2.2	• declare the size of one-dimensional arrays; for example: A[1:n] • show understanding of the use of the index variables in arrays • read values into an array using a FOR … TO … NEXT loop	Learners write programs from algorithms developed in unit 7, to read values from a data source (file, data statement, keyboard) into a specified array and calculate e.g. average, largest, smallest. These will have been tailored to give appropriate challenge to learners. **(G)/(I)**	Notes on arrays (as for Unit 7): **www.teach-ict.com/gcse_computing/ocr/216_programming/handling_data/miniweb/pg10.htm**

Appendix D: Sample lesson plan 1

Computer architecture and the fetch execute cycle

Lesson: Computer architecture and the fetch-execute cycle	**School:**	
Date:	**Teacher name:**	
Class:	**Number present:**	**Absent:**

Teaching Aims	• show what happens during the fetch part of the cycle • show what happens during the execute part of the cycle
Lesson objectives	• develop understanding of registers and their uses • develop understanding of the stages of the fetch-execute cycle
Syllabus assessment objectives	• describe the stages of the fetch-execute cycle, including the use of registers
Vocabulary, terminology and phrases	Program counter (PC) Memory Address Register (MAR) Memory Buffer Register (MBR) Current Instruction Register (CIR)
Previous learning	• understanding of the basic Von Neumann model for a computer system and the stored program concept (program instructions and data are stored in main memory and instructions are fetched and executed one after another)
Anticipated learner problems	• some learners may not understand the previous learning as it is one of the more challenging parts of the syllabus
Solutions to the problems	• use group work to reinforce the principles • recap and demonstrate using Little Man Computer
Resources	• http://gcsecomputing.org.uk/lmc/lmc.html • Diagram used for last lesson showing Von Neumann architecture but with names of registers removed • Cards with stages of the fetch-execute cycle for group work • Laminated cards to write values on for register role play

Plan lesson content	
Planned timings	**Planned activities**
10 minutes	Brief recap of the basic Von Neumann model for a computer system and the stored program concept. Revision of register names and their purposes.
10 minutes	Demonstration of the fetch-execute cycle with a short program. Using http://gcsecomputing.org.uk/lmc/lmc.html or similar.
10 minutes	Working in pairs work through a similar short program. Using http://gcsecomputing.org.uk/lmc/lmc.html or similar.
10 minutes	Working in small groups, arrange cards in the correct order for the fetch execute cycle.
10 minutes	Working individually, work through one/both of the programs writing down the contents of the registers.
Optional if time permits	Role play of fetch-execute cycle for a short program, with some learners taking the parts of the registers.

Plan consolidation	
Planned timings	**Planned activities**
10 minutes	Quiz, on register names and uses.

Plan assessment	
Topic	**Teacher Notes**
Peer or self-assessment	Self-assessment after quiz
Homework	Provision of a short program to follow the changes made to specific registers with, for example CIR and MAR, during the fetch-execute cycle
Exam question	Explain, with examples, what the PC and the MBR are used for in the fetch-execute cycle

Additional information	
Differentiation – how do you plan to give more support? How do you plan to challenge the more able learners?	
Reflection and evaluation	
Reflection Were the lesson objectives realistic? What did the learners learn today? What was the learning atmosphere like? Did my planned differentiation work well? Did I keep to timings? What changes did I make from my plan and why?	**Use the space below to reflect on your lesson. Answer the most relevant questions from the box on the left about your lesson.**
Summary evaluation	
What two things went really well (consider both teaching and learning)? 1: 2: **What two things would have improved the lesson (consider both teaching and learning)?** 1: 2: **What have I learned from this lesson about the class or individuals that will inform my next lesson?**	

Appendix E: Sample lesson plan 2

Programming concepts

Lesson: Computer architecture and the fetch-execute cycle	**School:**	
Date:	**Teacher name:**	
Class:	**Number present:**	**Absent:**

Teaching Aims	• introduce programming ideas in a fun, visual way • learners to have a positive experience of program development
Lesson objectives	• develop understanding of programming concepts • develop practical use of programming concepts
Syllabus assessment objectives	• understand and use the concepts of sequence, selection, repetition, totalling and counting
Vocabulary, terminology and phrases	sequence selection repetition totaling counting
Previous learning	• understand and use pseudocode
Anticipated learner problems	• some learners may have attempted programming before, others may be new to the concept
Solutions to the problems	• use of Scratch to introduce idea of program • working in pairs to start with
Resources	• Scratch http://scratch.mit.edu/ Some simple tasks in Scratch: http://www.teach-ict.com/programming/scratch/scratch_home.htm

Plan lesson content	
Planned timings	**Planned activities**
10 minutes	Brief recap of the concepts of sequence, selection, repetition, totalling and counting.
10 minutes	Demonstration of sequence, selection, repetition, totalling and counting, using a pre-prepared Scratch program.
10 minutes	Working in pairs to load and run the program, then change some values. For example, the number of repeats.
10 minutes	Working in pairs, develop and test a similar program.
10 minutes	Working individually, identify which instructions perform each action.
Optional if time permits	Demonstration of individual programs to the class.

Plan consolidation	
Planned timings	**Planned activities**
10 minutes	Individuals developing a new program from a given task

Plan assessment	
Topic	**Teacher Notes**
Peer or self-assessment	Self-assessment after program writing.
Homework	Provide **two** examples of sequence, selection, repetition, totalling and counting using Scratch.
Exam question	Explain, using examples, the difference between totalling and counting.

Additional information	
Differentiation – how do you plan to give more support? How do you plan to challenge the more able learners?	
Reflection and evaluation	
Reflection Were the lesson objectives realistic? What did the learners learn today? What was the learning atmosphere like? Did my planned differentiation work well? Did I keep to timings? What changes did I make from my plan and why?	**Use the space below to reflect on your lesson. Answer the most relevant questions from the box on the left about your lesson.**
Summary evaluation	
What two things went really well (consider both teaching and learning)? 1: 2: **What two things would have improved the lesson (consider both teaching and learning)?** 1: 2: **What have I learned from this lesson about the class or individuals that will inform my next lesson?**	

Appendix F: Sample lesson plan template

Lesson topic

Lesson:	**School:**	
Date:	**Teacher name:**	
Class:	**Number present:**	**Absent:**
Teaching Aims		
Lesson objectives		
Syllabus assessment objectives		
Vocabulary, terminology and phrases		
Previous learning		
Anticipated learner problems		
Solutions to the problems		
Resources		

Plan lesson content	
Planned timings	**Planned activities**

Plan consolidation	
Planned timings	**Planned activities**

Plan assessment	
Topic	**Teacher Notes**
Peer or self-assessment	
Homework	
Exam question	

Additional information	
Differentiation – how do you plan to give more support? How do you plan to challenge the more able learners?	
Reflection and evaluation	
Reflection Were the lesson objectives realistic? What did the learners learn today? What was the learning atmosphere like? Did my planned differentiation work well? Did I keep to timings? What changes did I make from my plan and why?	**Use the space below to reflect on your lesson. Answer the most relevant questions from the box on the left about your lesson.**
Summary evaluation	
What two things went really well (consider both teaching and learning)? 1: 2: **What two things would have improved the lesson (consider both teaching and learning)?** 1: 2: **What have I learned from this lesson about the class or individuals that will inform my next lesson?**	

Cambridge International Examinations
1 Hills Road, Cambridge, CB1 2EU, United Kingdom
tel: +44 1223 553554 fax: +44 1223 553558
email: info@cie.org.uk www.cie.org.uk