UNIT 1: PRINCIPLES OF COMPUTER SCIENCE

Delivery guidance

Approaching the unit

This unit allows your learners to explore the ways in which the features of computer programming can be applied to solve problems. The focus should be on learners understanding how to deconstruct problems and develop a sound understanding of programming concepts. Learners will need to be able to plan solutions using pseudocode and flow charts, as well as develop the quality and functionality of given pseudocode or flow charts through applying common programming structures. They will also need to analyse how data is handled within a computer system, and how different programming paradigms can be applied.

There are many ways to solve problems and you should expose your learners to a wide range of examples of effective and less effective programming solutions. They should be able to interpret, use, debug and suggest improvements to code that has been given to them in any of the four programming and mark-up languages listed in the specification.

This delivery guide does not cover everything that needs to be delivered for completion of this unit but gives examples of delivery methods. You should refer to the specification for full details of all the content that needs to be covered.

In the external assessment, learners may be tested on any of the content in the specification. Learners must practise the correct and full completion of any templates given in sample assessment materials (SAMs) or past papers in preparation for their external assessment task.

Delivering the topics

For topic A, you could start by exploring the concepts of computational thinking. This could be done through class and small group discussions about how these concepts are applied to the process of problem-solving and computer programming. Small group activities could examine how to apply computational thinking skills to a given concept or problem. After your initial explanation of the skills and processes, learners could be given a simple/common computer game and asked to explore how computational thinking might apply to it.

You should only spend a short amount of classroom time introducing and explicitly teaching computational thinking as the emphasis should be very much on hands-on practical activities through which learners develop skills by experimentation and regular practice. Learners will practice applying computational thinking on numerous occasions while completing this qualification. This will also be a useful skill for them in further study or employment in the computing industry.

For topic B, learners would benefit from exposure to a wide range of different problems and scenarios for which they must write sections of pseudocode to describe the computing processes required. It would benefit them to start with simple one- or two-line processes and individual logical concepts, and steadily

build up the complexity of the solutions that learners are expected to develop. For example, to develop your learners' understanding of IF statements, you may ask them to write standalone IF statements to describe given conditions in a range of contexts, before they apply them to a larger structure.

Learners should be able to use pseudocode and flow charts to demonstrate their ability to develop computer programming solutions and to present the logic and structure in a way that could be implemented in an appropriate programming language. Give learners the opportunity to explore, evaluate and develop given pseudocode and flow charts in addition to the time that they spend independently developing a solution to a given problem. Pseudocode should be language-agnostic (independent of any programming language), and learners should use it to demonstrate their understanding of accepted computing concepts, eg consistent use of variable names, assignment of data types and sensible logical structures. Learners should present their flow charts using standard British Computer Society (BCS) symbols.

For topic C, give learners the opportunity to explore and apply standard programming structures and conventions in a range of contexts. Learners must demonstrate an understanding of common functions and data-handling processes. They also need to be able to apply logical and mathematical concepts and make use of common built-in functions of programming languages, validation techniques, and standard algorithms to solve problems. Learners should be able to apply these skills and techniques to given scenarios.

It would be beneficial for learners to explore, use and develop the content for this topic in a range of contexts. They should apply these effectively when writing pseudocode and explore their use in the given programming and mark-up languages. Learners will not be expected to write large sections of computer code from scratch, but they should be able to identify and analyse the use of sections of code, suggesting improvements and changes as appropriate. You should supply opportunities for learners to explore good and not-so-good examples of the structures and processes, identified both as pseudocode and as computer-based code. When exploring code, give learners the opportunity to explore examples in hard copy (as would be expected in the exam) and also within a coding or development environment. This will allow them to see the outcomes of effective and less effective programming techniques, as well as the effect of changes.

For topic D, learners should explore different programming paradigms and how they are used to solve a range of problems. When delivering this content, you could start with activities that focus on what a particular paradigm is used for, using the given languages as a context. Ensure that learners understand the concepts and principles behind each particular paradigm and language. After exploring the uses and implications of a particular language, start to expose learners to actual code that follows a specific paradigm. Learners should be given the opportunity to explore, change and develop the given code in order to see the effects of effective and less effective code.

Exploring code within each paradigm will also allow learners to develop their understanding of concepts studied in earlier topics. Where possible, highlight these connections to support learners' progression. As learners are exposed to additional paradigms, you could give them follow-up activities, such as asking learners to consider how a given piece of code in one language might be translated into another and the implications of implementing the code in a different language. Learners should also be able to communicate their understanding of what a given piece of code would do (within a specific context), including how data would be processed by the program, the efficiencies and inefficiencies in the program, and ways in which the code could be improved.

978-1-44692-831-8 – Pearson BTEC Nationals in Computing – Delivery Guide
Issue 1 –July © Pearson 2016

High quality, accurate verbal and written communication skills are vital for progression into higher education and in employment. As such, learners should be confident in presenting thoughts and ideas to others, as well as producing well-presented, accurate and appropriate documentation for all stages of a project. Learners must be able to effectively evaluate the success of a project and the factors that contributed to the final outcome, including their own skills, knowledge and behaviours.

Assessment guidance

Learners will be assessed by a written examination paper, including short-answer questions, extended tasks and tasks requiring diagrammatical explanations and solutions. This will assess both their computational thinking skills and their understanding of the principles of computer science.

In preparation for the examination, learners should develop solutions to given problems using pseudocode and flow charts. They should be able to analyse and evaluate the efficiency of given solutions in the form of pseudocode, flow charts and code, as well as able to explore the implications of different solutions in given contexts. The practical application of these skills should be supported by the development of exam techniques, such as how to identify the requirements of specific command words and how to structure and present answers.

The examination will be presented as an examination paper and will be supported by an information booklet containing guidance on unusual or non-standard pseudocode, code functions, etc, that have been used in the examples. The booklet may also contain additional materials such as code extracts and planning documents that will be required to answer the questions.

For full details of assessment, refer to the SAMs and the specification.

Getting started

This gives you a starting place for one way of delivering the unit. Activities are suggested in preparation for the external assessment.

Unit 1: Principles of Computer Science

Introduction

Within any area of the computing industry, people will need to be able to identify aspects of a problem, analyse the needs of a user and/or client, and supply a suitable solution that meets the identified needs. This unit should give learners a solid understanding of the key principles of computer programming and equip them with the skills and knowledge required to deconstruct problems and develop effective computing solutions. These transferable skills will equip learners for further study or employment in the computing industry.

Topic A – Computational thinking

- You could begin by introducing the aim of the unit (ie the basic principles of computer programming and problem-solving) and how the concepts of computational thinking can be applied to practical problems.
- Explain to learners that computational thinking can be thought about in terms of four 'stages': decomposition, pattern recognition, pattern generalisation and abstraction, and algorithm design.
- Learners could work in small groups to look at an example of a program (such as a simple computer game) and identify how each of the four 'stages' of computational thinking may be applied. For example:
  - for 'decomposition', learners might be expected to explain the stages of the game, the win scenario, the role of a particular character in the game etc
  - for 'pattern recognition', they may identify that certain non-playable characters move in the same way
  - for 'pattern generalisation and abstraction', they could identify the required variables or start defining the required inputs
  - for 'algorithm design', they could describe parts of the game, such as what happens when a playable character makes contact with a certain item, in order to plan the logical steps needed in readiness for writing code.
- Learners should understand that computational thinking is not specific knowledge to be recalled but a set of skills and a way of working that can be applied to aspects of this and other units.
- Give learners the opportunity to explore a range of problems to which they can apply their computational thinking skills. However, after an initial introduction and some individual and group activities that apply to specific examples, learners should continue to develop these skills while working on the subsequent topics in this unit.
- Examples that learners could be given to work with include:
  - the invoice problem – applying different discounts to different products and then applying VAT to only part of an invoice because some goods are zero-rated
  - the commercial greenhouse where the environment is computer controlled.

978-1-44692-831-8 – Pearson BTEC Nationals in Computing – Delivery Guide
Issue 1 –July © Pearson 2016

Topic B – Standard methods and techniques used to develop algorithms

This topic is designed to allow learners to start to develop an understanding of the need for logical structures within computer programs. Ensure that they have plenty of opportunities to explore how these structures can be applied to solve problems.

- The introduction to this topic should connect logically to the previous topic. For example, you could introduce pseudocode as a natural development of 'algorithm design' covered in the previous topic.
- To develop learners' understanding, it may be beneficial for them to start by looking at the sequencing of instructions, as this will allow them to explore the importance of identifying the correct order in which tasks should be carried out. Learners could work on this individually, or in small groups, to look at given problems. You could also give them examples of solutions that do not always give tasks in the correct order, as this would allow them to identify the problem with the proposed solution and consider its potential impact.
- Give learners pseudocode that has instructions in the wrong order, for example VAT calculated on an invoice total rather than parts of an invoice, or invoice lines because some of the items are zero-rated in terms of VAT.

  This particular problem is not only about the sequence of instructions, but also about understanding the nature of the problem.
- Learners should then explore the logical structures that can be used to describe computer processes such as IF statements and loops (see the specification for a full list of processes). You could give learners sample code and ask them to explain what the statements in the code do. They should then move on to using pseudocode to describe identified processes before deconstructing a problem and describing a solution with pseudocode.

  There are many examples that could be given here including:

  - the environmental controls in the commercial green house will all be activated based on sensor readings and decisions made such as humidity, temperature, feeding cycles, watering cycles (both scheduled and responsive).

  The example above could be distributed to smaller groups in the class, each taking part of the problem.
- Depending on your learners' previous experience with computer programming, you may wish to limit initial pseudocode writing tasks to a few lines describing a single process or action. You can increase the length and complexity of the tasks as learners' skills develop.
- Vary the level of scaffolding you give for the devised tasks. Sometimes you may wish to give learners a large amount of pseudocode, which they may have to evaluate and debug. At other times, you may wish to give them a scenario and some success criteria for which they must devise an algorithm. This will help learners to develop the ability to apply their skills in context.
- Learners must be able to use and understand flow charts that make use of standard British Computer Society (BCS) symbols. As with pseudocode, learners must be able to interpret and evaluate solutions presented in the form of flow charts, as well as to present their own ideas using the correct symbols and structures.
- Explain that pseudocode should be language-agnostic. Learners will use it as a vehicle to demonstrate their understanding of accepted computing concepts, eg order of instructions, use of naming conventions for variables and efficient structures.

Topic C – Programming paradigms

This topic gives you a natural link between topic B and topic D. It gives learners the opportunity to explore the concepts and ideas that are common to many programming languages.

- You will have mentioned some of the concepts in this topic during the previous topic – for example, basic arithmetic operations would have been used when writing pseudocode algorithms. Therefore, some of the concepts in this unit could be introduced by revisiting a scenario and algorithm (or sets of algorithms) previously developed. You could then set activities based on these previous scenarios. These activities should allow learners to consider concepts that may not have been introduced before, such as considering the data types to be used. This would also allow learners to explore the reasons why particular data types might be used and how they affect the development of program and its functionality.

  For example, one of the key issues with data types is that there pre-written functions and methods in most languages that provide common basic actions linked to an object's data type. Consider the string data type:

  - finding the length of a string
  - placing all characters in upper or lower case (because an upper case 'A' is not the same as a lower case 'a' as they have different ASCII codes
  - reversing a string
  - comparing strings.

  These functions and methods only work on objects that are strings.

- Revisiting algorithms will give you further opportunities to introduce new programming concepts. If learners have already been asked to develop a number of algorithms in pseudocode (for example, for a fitness-tracking mobile app), use this example to build upon their skills. In this example, the developed algorithms may describe the processes of calculating body mass index (BMI), calculating kilocalories burned and adding together the number of kilocalories used over a given period. Ask learners to consider how they could integrate these algorithms into a larger program and to consider the impact that the algorithms in their current form would have on a larger program. This would allow you to explore factors such as modularity, global versus local variables and the ways in which values can be passed from one 'sub-routine' to another.
- Introduce the concept of large scale business systems development. For example, developing a CRM system for a business, which could be implemented using a phased strategy (by functional business area, adding new functionality once developed functionality is confirmed as working). Cover how risks could be mitigated, such as running old and new systems in parallel for a period of time (which creates more work but gives business continuity if the new system fails).
- Learners should also understand that they should create functions for actions they cannot find in libraries but might reuse - for example a validation routine to validate a number in a range. The parameters passed in would include the upper and lower limit for the number, plus the number itself, and the function would output a Boolean indicating the number is or is not valid.
- In this topic, learners are likely to work mostly with pseudocode as a way of exploring the concepts without the additional demand of learning the syntax of a particular language. As learners start to understand the concepts covered in the topic, you could start to show them how you would execute these concepts within a programming environment, using one of the programming languages covered in topic D. Python 3.4 (or subsequent version) would give a good starting point for many learners as its structure gives them a natural link between pseudocode and

978-1-44692-831-8 – Pearson BTEC Nationals in Computing – Delivery Guide
Issue 1 –July © Pearson 2016

programming. Python also allows users to execute very small amounts of code.

- Learners must know how a number of common algorithms work, as well as why and how to use them. You could give examples of these standard algorithms and, as learners develop their understanding, you should start to expect them to apply these algorithms in their own solutions.
- Throughout this topic, you should expose learners to a wide range of scenarios, giving varying levels of support as they complete the given tasks. These scenarios should reflect the types of tasks that may appear in the examination.

  Typical tasks that help to develop programming skills include:

  - the cash point
  - conversion programs
  - displaying multiplication tables
  - guessing games
  - sorting algorithms.

Topic D – Types of programming and mark-up language

By the end of this topic, learners will need to understand the features, applications and implications of four different languages: C family, Visual Basic, HTML5 (or subsequent version) and Python 3.4 (or subsequent version).

- When delivering this topic, you may wish to start with activities that focus on why a particular language may be used and the concepts and principles behind using it to create programs and other digital content.
- After exploring the uses of one particular language and the implications of the choice of language, expose learners to actual code that follows a specific paradigm. Give learners the opportunity to explore, change and develop the given code in order to see the results of effective (and not so effective) code.
- If you used a particular programming language as an example in topic C, you may want to choose this in order to give learners an easy transition between the two topics. Ensure that learners have a solid understanding of each language before moving on to the next. Learners will not have to write large sections of code from scratch, but they will need a working knowledge of each of the four language families listed in the specification, so that they can analyse, evaluate, debug and develop given code in the examination.
- Give learners the opportunity to experiment with code within each paradigm, allowing them to make changes and additions to see how the code works. This will develop learners' understanding of the concepts studied in the earlier topics.
- You could give learners follow-up activities, such as asking them to compare how a given piece of code in one programming language might be translated into another and the subsequent implications of implementing the code in a different language. Learners should also be able to communicate their understanding of what a given piece of code would do within a given scenario. They should consider how the program would process data, efficiencies and inefficiencies in the program, and ways in which the code could be improved.

Details of links to other BTEC units and qualifications, and to other relevant units/qualifications

Pearson BTEC Level 3 Nationals in Computing (NQF):

- *Unit 3: Planning and Management of Computer Projects*
- *Unit 4: Software Design and Development Project*
- *Unit 9: The Impact of Computing*

Resources

In addition to the resources listed below, publishers are likely to produce Pearson-endorsed textbooks that support this unit of the BTEC Nationals in Computing. Check the Pearson website (http://qualifications.pearson.com/en/support/published-resources.html) for more information as titles achieve endorsement.

Websites

- www.python.org/
  Python - an open-source programming language. The website contains downloads for various operating systems and official documentation
- www.w3schools.com/html/default.asp
  W3Schools – offers tutorials in web development languages, including HTML (with sections on HTML5), covering basic and more complex features.
- https://msdn.microsoft.com/en-us/library/2x7h1hfk.aspx
  Microsoft® Developer Network – guidance for using Visual Basic®, including language walkthroughs.
- www.microsoftvirtualacademy.com/en-us/training-courses/vb-fundamentals-for-absolute-beginners-8297
  Microsoft Virtual Academy – a series of introductory tutorial videos on using Visual Basic.

978-1-44692-831-8 – Pearson BTEC Nationals in Computing – Delivery Guide
Issue 1 –July © Pearson 2016