AQA A-level

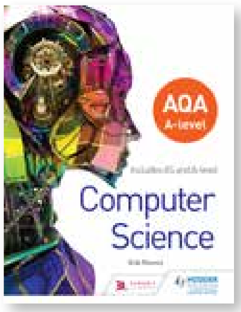
Includes AS and A-level

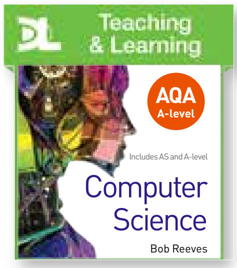
Computer Science

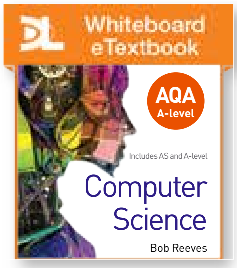
Bob Reeves

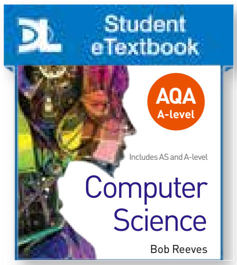
DYNAMIC LEARNING

HODDER EDUCATION LEARN MORE









Contents

Introduction

What is computer science?

The world of computer science continues to develop at an amazing rate. If you had spoken to an A-level student embarking on a computer science course just ten years ago they might not have believed that in the year 2015 we would all be permanently connected to the internet on smart phones, watching movies in high definition on 55-inch curved-screen TVs, streaming our favourite music to our phones from a database of millions of tracks stored in "the cloud" or carrying round a tablet that has more processing power than the flight computer on the now decommissioned space shuttle.

No-one really knows where the next ten years will take us. The challenge for you as a computer scientist is to be able to respond to this ever-changing world and to develop the knowledge and skills that will help you to understand technology that hasn't yet been invented!

Studying A-level computer science gives you a solid foundation in the underlying principles of computing, for example: understanding how algorithms and computer code are written; how data is stored; how data is transmitted around networks; and how hardware and software work. It also provides you with a deeper level of understanding that goes beyond the actual technology. For example, you will learn about how to use computation to solve problems and about the close links between computer science, mathematics and physics.

You might be surprised to learn that many of the key principles of computing were developed before the modern computer, with some concepts going back to the ancient Greeks. At the same time, you will be learning about the latest methods for solving computable problems in today's world and developing your own solutions in the form of programs or apps.

Studying computer science at A-level is challenging, but it is also highly rewarding. There are very few jobs that do not involve the use of computers and having a good understanding of the science behind them will effectively prepare you for further study or employment.

Course coverage and how to use this book

This book has been written to provide complete coverage of the AQA Computer Science specifications for AS and A-level that are taught from September 2015. The content of the book is matched and sequenced according to the specification, and organised into sections in accordance with the main specification headings used by AQA.

Students studying A-level need to be familiar with all of the content of the AS specification and in addition need to cover those sections highlighted throughout the text, which are unique to A-level. There is support for every section of the specification including the written papers, and general advice on tackling the skeleton program and the coursework.

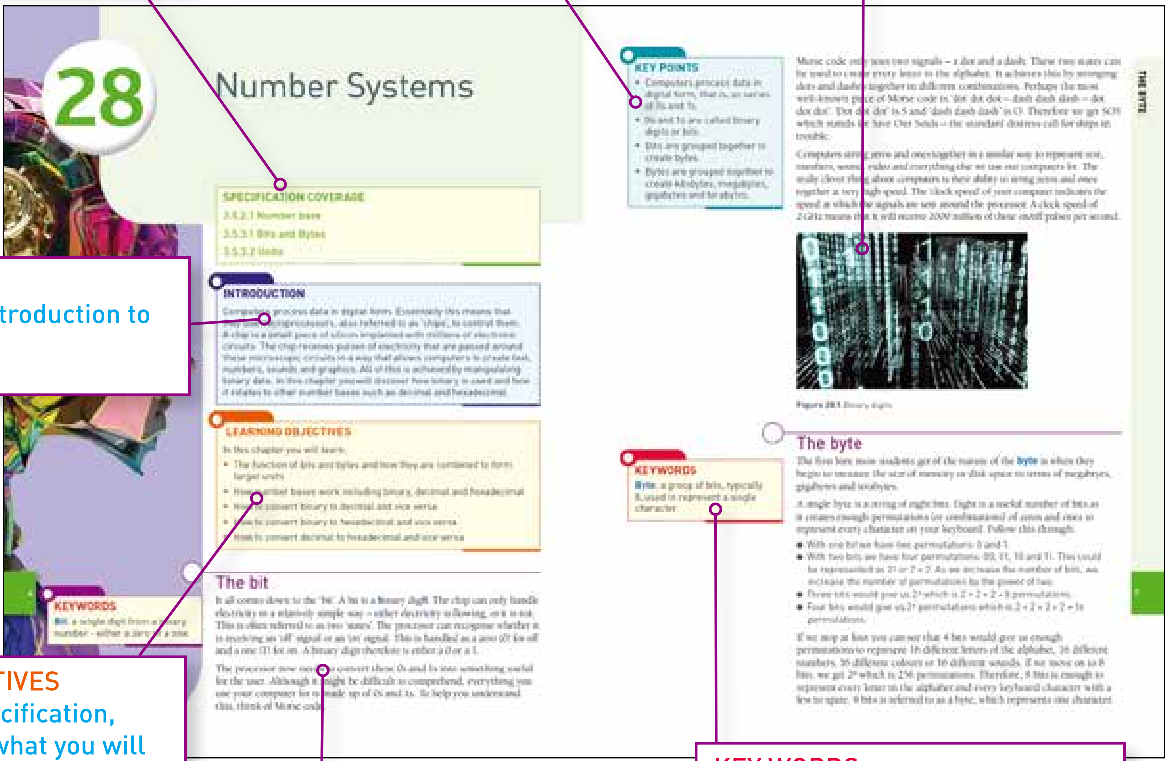
The main objective of the book is to provide a solid foundation in the theoretical aspects of the course. Further support and practical examples of coded solutions are provided on line via Dynamic Learning.

Chapters contain:

Specification coverage
Taken directly from the specification, it shows which elements of AS and A-level are covered within each chapter.

KEY POINTS
All of the main points for each chapter are summarised. These are particularly useful as a revision aid.

Diagrams and images
The book uses diagrams and images wherever possible to aid understanding of the key points.



INTRODUCTION
This is a concise introduction to set the scene.

LEARNING OBJECTIVES
Matched to the specification, these summarise what you will learn by the end of the chapter.

The main text
This contains detailed definitions, explanations and examples.

KEY WORDS
All of the key words are identified with concise definitions. These form a glossary, which is useful for revision and to check understanding.

Acknowledgements

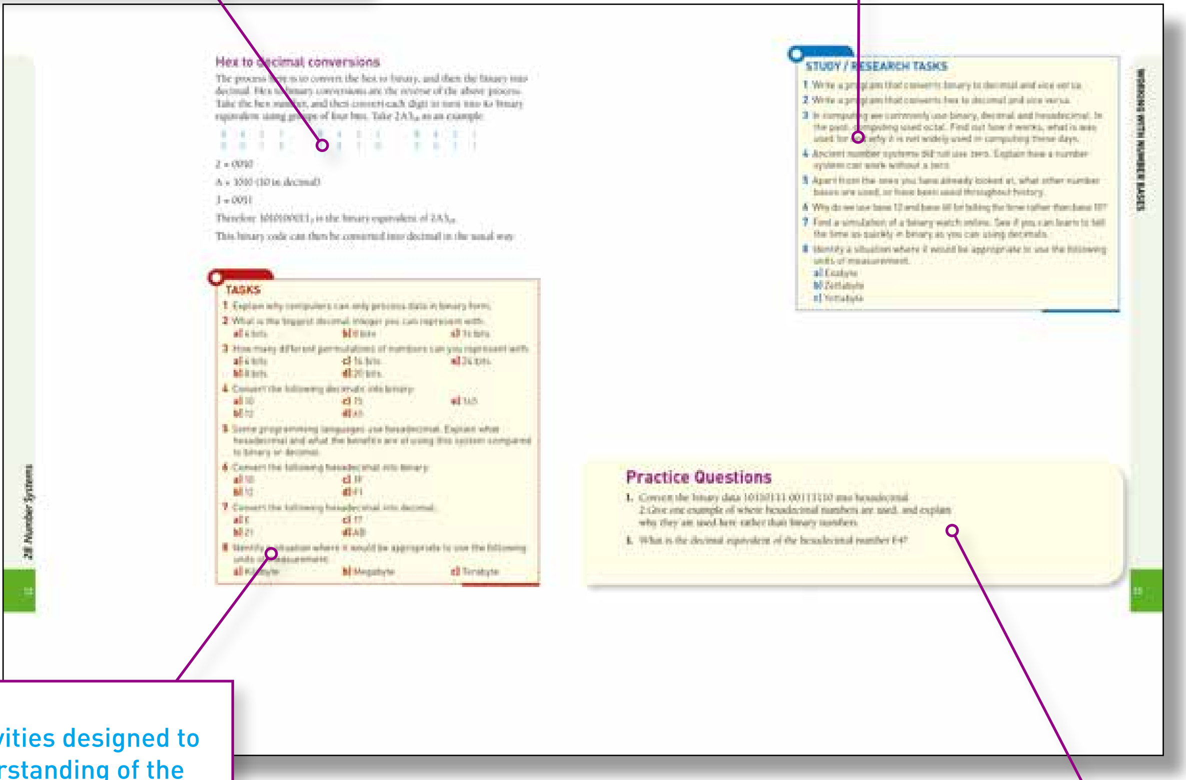
Dave Fogg for producing the code examples used.

Paul Varey for his initial proofread.

Dedicated to Eli Reeves

Code example
Where relevant there are examples of pseudo-code or actual code to demonstrate particular concepts. Code examples in this book are mainly written using the VB.NET framework. VB Express 10.0 has been used as this is available as a free download.
The code can also be migrated into other versions of VB.

RESEARCH/STUDY QUESTIONS
These questions go beyond the specification and provide a further challenge designed to encourage you to "read around the subject" or develop your skills and knowledge further.



Hex to decimal conversions

[illegible]

STUDY / RESEARCH TASKS

[illegible]

TASKS

[illegible]

Practice Questions

[illegible]

TASKS
These are activities designed to test your understanding of the contents of the chapter. These may be written exercises or computer tasks.

Practice questions
These are provided for each section and are contextualised so that they match the style of AQA questions.

28 Number Systems

SPECIFICATION COVERAGE

3.5.2.1 Number base

3.5.3.1 Bits and Bytes

3.5.3.2 Units

INTRODUCTION

Computers process data in digital form. Essentially this means that they use microprocessors, also referred to as 'chips', to control them. A chip is a small piece of silicon implanted with millions of electronic circuits. The chip receives pulses of electricity that are passed around these microscopic circuits in a way that allows computers to create text, numbers, sounds and graphics. All of this is achieved by manipulating binary data. In this chapter you will discover how binary is used and how it relates to other number bases such as decimal and hexadecimal.

LEARNING OBJECTIVES

In this chapter you will learn:

- The function of bits and bytes and how they are combined to form larger units
- How number bases work including binary, decimal and hexadecimal
- How to convert binary to decimal and vice versa
- How to convert binary to hexadecimal and vice versa
- How to convert decimal to hexadecimal and vice versa

KEYWORDS

Bit: a single digit from a binary number - either a zero or a one.

The bit

It all comes down to the 'bit'. A bit is a **b**inary dig**it**. The chip can only handle electricity in a relatively simple way – either electricity is flowing, or it is not. This is often referred to as two 'states'. The processor can recognise whether it is receiving an 'off' signal or an 'on' signal. This is handled as a zero (0) for off and a one (1) for on. A binary digit therefore is either a 0 or a 1.

The processor now needs to convert these 0s and 1s into something useful for the user. Although it might be difficult to comprehend, everything you use your computer for is made up of 0s and 1s. To help you understand this, think of Morse code.

KEY POINTS

- Computers process data in digital form, that is, as series of 0s and 1s.
- 0s and 1s are called binary digits or bits.
- Bits are grouped together to create bytes.
- Bytes are grouped together to create kilobytes, megabytes, gigabytes and terabytes.

Morse code only uses two signals – a dot and a dash. These two states can be used to create every letter in the alphabet. It achieves this by stringing dots and dashes together in different combinations. Perhaps the most well-known piece of Morse code is 'dot dot dot – dash dash dash – dot dot dot'. 'Dot dot dot' is S and 'dash dash dash' is O. Therefore we get SOS which stands for Save Our Souls – the standard distress call for ships in trouble.

Computers string zeros and ones together in a similar way to represent text, numbers, sound, video and everything else we use our computers for. The really clever thing about computers is their ability to string zeros and ones together at very high speed. The 'clock speed' of your computer indicates the speed at which the signals are sent around the processor. A clock speed of 2 GHz means that it will receive 2000 million of these on/off pulses per second.

Figure 28.1 Binary digits

The byte

KEYWORDS

Byte: a group of bits, typically 8, used to represent a single character.

The first hint most students get of the nature of the **byte** is when they begin to measure the size of memory or disk space in terms of megabtyes, gigabytes and terabytes.

A single byte is a string of eight bits. Eight is a useful number of bits as it creates enough permutations (or combinations) of zeros and ones to represent every character on your keyboard. Follow this through:

- With one bit we have two permutations: 0 and 1.
- With two bits we have four permutations: 00, 01, 10 and 11. This could be represented as 2^2 or 2 × 2. As we increase the number of bits, we increase the number of permutations by the power of two.
- Three bits would give us 2^3 which is 2 × 2 × 2 = 8 permutations.
- Four bits would give us 2^4 permutations which is 2 × 2 × 2 × 2 = 16 permutations.

If we stop at four you can see that 4 bits would give us enough permutations to represent 16 different letters of the alphabet, 16 different numbers, 16 different colours or 16 different sounds. If we move on to 8 bits, we get 2^8 which is 256 permutations. Therefore, 8 bits is enough to represent every letter in the alphabet and every keyboard character with a few to spare. 8 bits is referred to as a byte, which represents one character.

The basic fact here is that the more bits you use, the greater the range of numbers, characters, sounds or colours that can be created. Taking numbers as an example, as we have seen, 8 bits would be enough to represent 256 different numbers (0 – 255). As the number of bits increases, the range of numbers increases rapidly. For example 2^{16} would give 65 536 permutations, 2^{24} would give approximately 16 million and 2^{32} would give over 4 billion.

Units

KEYWORDS

Unit: The grouping together of bits of bytes to form larger blocks of measurement e.g. KG, MB.

Larger combinations of bytes are used to measure the capacity of memory and storage devices. The size of the **units** can be referred to either using binary or decimal prefixes. For example, in decimal, the term kilo is commonly used to indicate a unit that is 1000 times larger than a single unit. So the correct term would be kilobyte (K). In binary, the correct term is actually kibibyte (Ki) with 1024 bytes being the nearest binary equivalent to 1000.

It is common practice to show the size of the numbers using superscript values. For example 2^{10} Ki indicates binary (base 2) to the power of ten, which is 1024 bytes. 10^3 K indicates decimal base ten to the power of three, which is 1000 bytes.

Common units are shown below using both binary and decimal prefixes:

Binary			Decimal		
kibibyte	Ki	2^{10}	kilobyte	K	10^3
mebibyte	Mi	2^{20}	megabyte	M	10^6
gibibyte	Gi	2^{30}	gigabyte	G	10^9
tebibyte	Ti	2^{40}	terabyte	T	10^{12}

Number bases

KEYWORDS

Number base: The amount of digits available within a particular number system, e.g. base 10 for decimal, base 2 for binary.

A **number base** indicates how many different digits are available when using a particular number system. For example, decimal is number base 10 which means that it uses ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 and binary is number base 2 which means that it uses two digits: 0 and 1. Different number bases are needed for different purposes. Humans use number base 10, whereas computers use binary as this represents digital data.

The number base determines how many digits are needed to represent a number. For example, the number 98 in decimal (base 10) requires two digits. The binary (base 2) equivalent is 1100010 which requires seven digits. As a consequence of this there are many occasions in computing when very long binary codes are needed. To solve this problem, other number bases can be used, which require fewer digits to represent numbers. For example, some aspects of computing involve number base 16 which is referred to as 'hexadecimal'.

The accepted method for representing different number bases (in textbooks and exam questions) is to show the number with the base in subscript. For example:

- 43_{10} is decimal
- 1011_2 is binary
- $2A7_{16}$ is hexadecimal.

KEY POINTS

- Computing uses three main number bases: binary (base 2), decimal (base 10) and hexadecimal (base 16).
- You need to be able to convert between the three number bases.

Hexadecimal

Hexadecimal or ‘hex’ is particularly useful for representing large numbers as fewer digits are required. Hex is used in a number of ways. Memory addresses are shown in hex format, as are colour codes. The main advantage of hex is that two hex digits represent one byte.

Consider the number 11010011_2. This is an 8-bit code which when converted to decimal equals 211_{10}. The same number is hex is $D3_{16}$. This basic example shows that an 8-bit code in binary can be represented as a two-digit code in hex. Consequently hex is often referred to as ‘shorthand’ for binary as it requires fewer digits.

As it is number base 16, hex uses 16 different digits: 0 to 9 and A to F. The table below shows decimal numbers up to 31 with the hex equivalents:

Decimal	Hex	Decimal	Hex
0	0	16	10
1	1	17	11
2	2	18	12
3	3	19	13
4	4	20	14
5	5	21	15
6	6	22	16
7	7	23	17
8	8	24	18
9	9	25	19
10	A	26	1A
11	B	27	1B
12	C	28	1C
13	D	29	1D
14	E	30	1E
15	F	31	1F

There is scope for confusion here as humans rarely use letters as numbers. Also, the numbers in hex may convert to different numbers in decimal. For example, the number 16 in decimal is the equivalent of the number 10 (one zero) in hex.

Working with number bases

When performing any calculations, humans use number base 10, probably because we have ten digits on our hands. Commonly this system is known as decimal and uses 10 different digits: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. When we get to 9 we add an extra digit and start again. When we get to 99, we add a further digit and so on. Each digit we add is worth ten times the previous digit. This is easier to understand if you think back to how you were taught maths at primary school.

The number 2098 is easy to understand in decimal terms. To state the obvious, it is made up of $(2 \times 1000) + (0 \times 100) + (9 \times 10) + (8 \times 1)$. When creating a number, we start with the units and add the further digits as needed to create the number we want. Each extra digit is ten times the previous one because we are using number base 10.

Binary is number base 2 and works on exactly the same principle. This time we only have two digits, 0 and 1. It has to be binary because computers only work by receiving a zero or one (off and on). So, 1 is the biggest number we can have with one bit. To increase the size of the number, we add more bits. Each bit is worth two times the previous bit because we are using number base 2. The table below shows an 8-bit binary number 10000111. Notice the value of each new bit is increasing by 2 each time, as binary is base 2.

128	64	32	16	8	4	2	1
1	0	0	0	0	1	1	1

Again, using the same principle as with decimal to work out the number we have:

(1 × 128) + (1 × 4) + (1 × 2) + (1 × 1). This adds up to 135.

Therefore 10000111 in binary = 135 in decimal.

Binary to decimal conversions

Binary numbers are converted to decimal integers as follows:

- Write down a binary number (e.g. 10000111).
- Above the number, starting from the 'least significant bit' (LSB) write the number 1.
- As you move left from the LSB to the 'most significant bit' (MSB) double the value of the previous number:

MSB							**LSB**
128	64	32	16	8	4	2	1
1	0	0	0	0	1	1	1

- Wherever there is a 1, add the decimal value: the above example represents one 128, one 4, one 2 and a 1 giving a total value of 135 (128 + 4 + 2 + 1 = 135). Therefore 10000111 in binary equals 135 as a decimal integer.

Decimal to binary conversions

To convert a decimal integer to a binary number, use the same method as above, but working the other way. For example, to convert the number 98:

- Write down the power of 2 sequence. (Eight bits are used here but you will notice that you only need seven for this example.)

MSB							**LSB**
128	64	32	16	8	4	2	1

- Starting from the MSB put a 1 or 0 in each column as necessary to ensure that it adds up to 98 as follows:
  0 under 128
  1 under 64
  1 under 32
  0 under 16
  0 under 8
  0 under 4
  1 under 2
  0 under 1

Therefore 98 in decimal = 01100010 in binary.

Another way of carrying out this calculation is to carry out repeated divisions on the decimal number as follows:

98 divided by 2 = 49 with a remainder of 0
49 divided by 2 = 24 with a remainder of 1
24 divided by 2 = 12 with a remainder of 0
12 divided by 2 = 6 with a remainder of 0
6 divided by 2 = 3 with a remainder of 0
3 divided by 2 = 1 with a remainder of 1
1 divided by 2 = 0 with a remainder of 1

Notice that you keep dividing by 2 until there is nothing left to divide. Reading from the bottom this gives us 1100010 which equals 98. (Note that the leading zero is omitted.)

Check your answer by working it back the other way:

MSB							**LSB**
128	64	32	16	8	4	2	1
0	1	1	0	0	0	1	0

64 + 32 + 2 = 98

Decimal to hex conversions

A common approach to convert decimal integers to hex is to first convert the decimal to binary and then convert the binary to hex. Taking the decimal number 211 as an example:

- Work out the binary equivalent.

128	64	32	16	8	4	2	1
1	1	0	1	0	0	1	1

- Split the binary number into two groups of four bits and convert each into the hex equivalent.

8	4	2	1	8	4	2	1
1	1	0	1	0	0	1	1

Therefore $11010001_2 = 211_{10}$

8 + 4 + 1 = D (the hex equivalent of 13)
and 2 + 1 = 3

Therefore $211_{10} = 11010011_2 = D3_{16}$

Hex to decimal conversions

The process here is to convert the hex to binary, and then the binary into decimal. Hex to binary conversions are the reverse of the above process. Take the hex number, and then convert each digit in turn into its binary equivalent using groups of four bits. Take $2A3_{16}$ as an example:

8	4	2	1	8	4	2	1	8	4	2	1
0	0	1	0	1	0	1	0	0	0	1	1

2 = 0010

A = 1010 (10 in decimal)

3 = 0011

Therefore 1010100011_2 is the binary equivalent of $2A3_{16}$.

This binary code can then be converted into decimal in the usual way:

TASKS

1 Explain why computers can only process data in binary form.

2 What is the biggest decimal integer you can represent with:
a) 4 bits **b)** 8 bits **c)** 16 bits

3 How many different permutations of numbers can you represent with:
a) 4 bits **b)** 8 bits **c)** 16 bits **d)** 20 bits **e)** 24 bits

4 Convert the following decimals into binary:
a) 10 **b)** 12 **c)** 15 **d)** 65 **e)** 165

5 Some programming languages use hexadecimal. Explain what hexadecimal and what the benefits are of using this system compared to binary or decimal.

6 Convert the following hexadecimal into binary:
a) 10 **b)** 12 **c)** 1F **d)** F1

7 Convert the following hexadecimal into decimal:
a) E **b)** 21 **c)** 17 **d)** AB

8 Identify a situation where it would be appropriate to use the following units of measurement:
a) Kilobyte **b)** Megabyte **c)** Terabyte

STUDY / RESEARCH TASKS

1 Write a program that converts binary to decimal and vice versa.

2 Write a program that converts hex to decimal and vice versa.

3 In computing we commonly use binary, decimal and hexadecimal. In the past, computing used octal. Find out how it works, what is was used for and why it is not widely used in computing these days.

4 Ancient number systems did not use zero. Explain how a number system can work without a zero.

5 Apart from the ones you have already looked at, what other number bases are used, or have been used throughout history.

6 Why do we use base 12 and base 60 for telling the time rather than base 10?

7 Find a simulation of a binary watch online. See if you can learn to tell the time as quickly in binary as you can using decimals.

8 Identify a situation where it would be appropriate to use the following units of measurement:
a) Exabyte
b) Zettabyte
c) Yottabyte

Practice Questions

1. Convert the binary data 10110111 00111110 into hexadecimal.
   2.Give one example of where hexadecimal numbers are used, and explain why they are used here rather than binary numbers.
3. What is the decimal equivalent of the hexadecimal number E4?

The Publishers would like to thank the following for permission to reproduce copyright material:

P.7 © carloscastilla – Fotolia.

© Bob Reeves 2015

First published in 2015 by

Hodder Education
An Hachette UK Company,
338 Euston Road
London NW1 3BH

Impression number 5 4 3 2 1

Year 2019 2018 2017 2016 2015

Cover photo © LaCozza – Fotolia

Typeset in 11/13 pt Berkely Oldstyle

ISBN 978 1 447 183951 1

AQA A-level Computer Science

Includes AS and A-level

This sample chapter is from AQA A-level Computer Science, the new textbook for the AQA AS and A-level specifications, for first teaching from September 2015.

This title has been selected for AQA's official approval process.

AQA A-level Computer Science gives students the confidence to think creatively and progress through the AQA AS and A-level specifications. Detailed coverage of the specifications enriches their understanding of the fundamental principles of computing, whilst a range of activities help to develop the programming and computational thinking skills they need for success at A-level and beyond.

- Helps build a thorough understanding of the fundamental principles examined in the AQA A-level Computer Science specifications (including programming, algorithms, data structures and representation, systems, databases and networks, uses and consequences)
- Provides clear coverage and progression through the AS and A-level specifications, written by a leading computer science author
- Prepares students to tackle the various demands of the course, from programming and theoretical assessments to the investigative project at A-level
- Helps students develop key skills through frequent coding and exam practice, in order that they can demonstrate and apply their knowledge of the principles of computer science, and design, program and evaluate problem-solving computer systems.

Bob Reeves is an experienced teacher and examiner, and well-respected author of resources for Computing and ICT across the curriculum.

Textbook subject to change based on Ofqual feedback

HODDER EDUCATION
www.hoddereducation.co.uk