CAMBRIDGE
International Education

Syllabus

Cambridge IGCSE™ (9–1)
Computer Science 0984

Use this syllabus for exams in 2029, 2030 and 2031.
Exams are available in the June and November series.

Version 1

For the purposes of screen readers, any mention in this document of Cambridge IGCSE refers to Cambridge International General Certificate of Secondary Education.

Cambridge Pathway

Why choose Cambridge?

We work with schools worldwide to build an education that shapes knowledge, understanding and skills. Together, we give learners the confidence they need to thrive and make a positive impact in a changing world.

As part of the University of Cambridge, we offer a globally trusted and flexible framework for education from age 3 to 19, informed by research, experience, and listening to educators.

With recognised qualifications, high-quality resources, comprehensive support and valuable insights, we help schools prepare every student for the opportunities and challenges ahead.

Qualifications that are recognised and valued worldwide

From the world's top-ranked universities to local higher education institutions, Cambridge qualifications open doors to a world of opportunities.

Setting a global standard

With over 160 years of experience in delivering fair, valid and reliable assessments to students worldwide, we offer a global, recognised performance standard for international education.

Your path, your way

Schools can adapt our curriculum, high-quality teaching and learning resources and flexible assessments to their local context. Our aligned offer helps Cambridge schools support every learner to reach their potential and thrive.

Learning with lasting impact

Cambridge learners build subject knowledge and conceptual understanding, and develop a broad range of skills, learning habits and attributes to help make them ready for the world.

Improving learning outcomes through data-led insight and action

Our trusted baseline and diagnostic assessments, together with our insights and evaluation service, help schools turn data into knowledge and actionable insights, to inform teaching decisions and improve learner outcomes.

Bringing together a community of experts

We bring together the collective knowledge of experts and our diverse community of educators worldwide, supporting them to learn from one another and share ideas and information.

Tackling the climate crisis together

We believe that education is key to tackling the climate crisis. Together with Cambridge schools, we can empower young people with the skills and knowledge to take action on climate change, helping them be ready for the world.

School feedback: 'We think the Cambridge curriculum is superb preparation for university.'

Feedback from: Christoph Guttentag, Dean of Undergraduate Admissions, Duke University, USA

Contents

Why choose Cambridge? **2**

1 Why choose this syllabus? **4**

2 Syllabus overview **7**

Aims 7

Content overview 8

Assessment overview 9

Assessment objectives 10

3 Subject content **11**

Computer Systems and Logic 11

Algorithms and Programming 22

4 Details of the assessment **28**

Paper 1 – Computer Systems and Logic 28

Paper 2 – Algorithms and Programming 28

Mathematical requirements 29

Flowchart symbols 29

Logic gate symbols 30

Python 3.10 guide 31

Command words 49

5 What else you need to know **50**

Before you start 50

Making entries 51

Accessibility and equality 52

After the exam 53

How students and teachers can use the grades 53

Changes to this syllabus for 2029, 2030 and 2031 54

Important: Changes to this syllabus

For information about changes to this syllabus for 2029, 2030 and 2031, go to page 54.

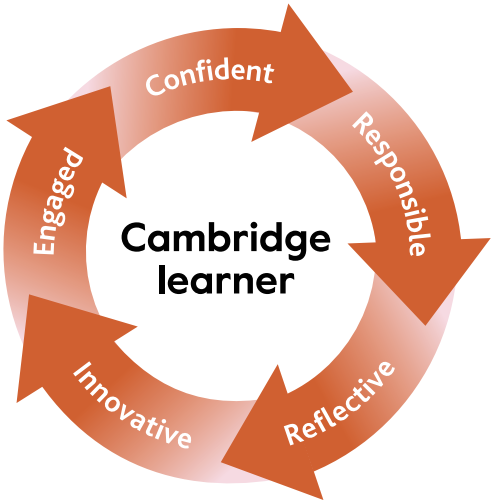
1 Why choose this syllabus?

Key benefits

Cambridge IGCSE is the world's most popular international qualification for 14 to 16 year olds, although it can be taken by students at any age. Taught by over 6000 schools in 150 countries, it is tried, tested and trusted.

Cambridge IGCSE offers a flexible curriculum, with a choice of over 70 subjects.

Our programmes promote a thorough knowledge and understanding of a subject and help to develop the skills learners need for their next steps in education or employment.



Cambridge IGCSE (9–1) Computer Science provides an ideal foundation in computer science. Learners gain confidence in computational thinking and programming, an appreciation of automated and emerging technologies and the benefits of their use. They develop an understanding of the main principles of problem-solving by creating computer-based solutions using algorithms and a high-level programming language. Learners also develop a range of technical skills including the ability to test effectively and to evaluate solutions.

Our approach in Cambridge IGCSE Computer Science encourages learners to be:

confident, interested in learning about computer science and using technical language to communicate their knowledge and understanding

responsible, working systematically, safely and securely when using technology

reflective, learning from their experiences when creating programs and using technology; understanding how technology impacts society

innovative, solving unfamiliar problems and designing computer programs creatively and independently

engaged, keen to develop computer science skills and further their understanding of developments in the use of technology.

School feedback: 'The strength of Cambridge IGCSE qualifications is internationally recognised and has provided an international pathway for our students to continue their studies around the world.'

Feedback from: Gary Tan, Head of Schools and CEO, Raffles Group of Schools, Indonesia

Qualifications that are recognised and valued worldwide

Cambridge qualifications prepare and equip learners with the skills they need to thrive at university and beyond. The world's best higher education institutions recognise our qualifications and value the critical thinking skills, independent research abilities and deep subject knowledge that Cambridge learners bring.

We continually work with universities and colleges in every part of the world to ensure that they understand and accept our qualifications. Cambridge IGCSE provides a springboard to the Cambridge Advanced stage, as well as other post-16 routes. The combination of knowledge and skills in Cambridge IGCSE (9–1) Computer Science gives learners a solid foundation for further study. Candidates who achieve grades 9 to 4 are well prepared to follow a wide range of courses including Cambridge International AS & A Level Computer Science.

Many universities require a combination of Cambridge International AS & A Levels and Cambridge IGCSEs or equivalent to meet their entry requirements.

UK ENIC, the national agency in the UK for the recognition and comparison of international qualifications and skills, has carried out an independent benchmarking study of Cambridge IGCSE and found it to be comparable to the standard of the GCSE in the UK. This means students can be confident that their Cambridge IGCSE qualifications are accepted as equivalent to UK GCSEs by leading universities worldwide.

Learn more at **www.cambridgeinternational.org/recognition**

School feedback: 'Cambridge IGCSE is one of the most sought-after and recognised qualifications in the world. It is very popular in Egypt because it provides the perfect preparation for success at advanced level programmes.'

Feedback from: Managing Director of British School of Egypt BSE

Supporting teachers

We believe education works best when teaching and learning are closely aligned to the curriculum, resources and assessment. Our high-quality teaching support helps to maximise teaching time and enables teachers to engage learners of all backgrounds and abilities.

We aim to provide the following support for each Cambridge qualification:

- Syllabus
- Specimen question papers and mark schemes
- Specimen paper answers
- Schemes of Work
- Example candidate responses
- Past papers and mark schemes
- Principal examiner reports for teachers.

These resources are available on the School Support Hub at **http://schoolsupporthub.cambridge.org**, our secure online site for Cambridge teachers. Your exams officer can provide you with a login.

Additional teaching and learning resources are also available for many syllabuses and vary according to the nature of the subject and the structure of the assessment of each syllabus. These can include ready-built lesson materials, digital resources and multimedia for the classroom and homework, guidance on assessment and much more. Beyond the resources available on the School Support Hub, a wide range of endorsed textbooks and associated teaching and learning support are available from Cambridge at **www.cambridge.org/education** and from other publishers. Resources vary according to the nature of the subject and the structure of the assessment of each syllabus.

You can also contact our global Cambridge community or talk to a senior examiner on our discussion forums.

Sign up for email notifications about changes to syllabuses, including new and revised products and services, at **www.cambridgeinternational.org/syllabusupdates**

Professional development

Find the next step on your professional development journey.

- **Introduction courses** – An introduction to Cambridge programmes and qualifications. For teachers who are new to Cambridge programmes or new to a specific syllabus.
- **Focus on Teaching courses** – These are for teachers who want to explore a specific area of teaching and learning within a syllabus or programme.
- **Focus on Assessment courses** – These are for teachers who want to understand the assessment of a syllabus in greater depth.
- **Marking workshops** – These workshops help you become more familiar with what examiners are looking for, and provide an opportunity to raise questions and share your experiences of the syllabus.
- **Enrichment Professional Development** – Transform your approach to teaching with our Enrichment workshops. Each workshop focuses on a specific area of teaching and learning practice.
- **Cambridge Professional Development Qualifications (PDQs)** – Practice-based programmes that transform professional learning for practising teachers. Available at Certificate and Diploma level.

For more information visit **www.cambridgeinternational.org/support-and-training-for-schools**

Supporting exams officers

We provide comprehensive support and guidance for all Cambridge exams officers.
Find out more at **www.cambridgeinternational.org/eoguide**

2 Syllabus overview

Aims

The aims describe the purposes of a course based on this syllabus.

Students following a course based on this syllabus will:

- build computational thinking skills for analysing and solving problems using computers
- apply their knowledge effectively to create solutions using Python
- understand the components of computer systems and how they work together
- understand the role of data transmission and integrity in communication, and apply principles of cybersecurity
- explore the development and use of automated and emerging technologies

We are an education organisation and politically neutral. The contents of this syllabus, examination papers and associated materials do not endorse any political view. We endeavour to treat all aspects of the exam process neutrally.

Content overview

Candidates study the following topics:

Computer systems and logic

1 Data representation and logic gates
2 Logic circuits and hardware
3 Data transmission and networking
4 Data integrity and cybersecurity
5 Software
6 Automated systems and robotics
7 Emerging technologies

Algorithms and programming

8 Programming fundamentals
9 Algorithm design
10 Further programming and testing
11 Databases and Structured Query Language (SQL)

Assessment overview

All candidates take two components. Candidates will be eligible for grades 9 to 1.

All candidates take:	**and:**
Paper 1 1 hour 45 minutes Computer Systems and Logic 50% 75 marks Short-answer and structured questions Questions will be based on Topics 1–7 of the subject content All questions are compulsory No calculators are permitted Externally assessed	**Paper 2** 1 hour 45 minutes Algorithms and Programming 50% 75 marks Short-answer and structured questions and a scenario-based question Questions will be based on Topics 8–11 of the subject content All questions are compulsory Calculators are permitted Externally assessed

Information on availability is in the **Before you start** section.

Assessment objectives

The assessment objectives (AOs) are:

AO1

Demonstrate knowledge and understanding of the principles and concepts of computer science.

AO2

Apply knowledge and understanding of the principles and concepts of computer science to a given context, including the analysis and design of computational or programming problems.

AO3

Provide solutions to problems by:

- evaluating computer systems
- making reasoned judgments
- presenting conclusions.

Weighting for assessment objectives

The approximate weightings allocated to each of the assessment objectives (AOs) are summarised below.

Assessment objectives as a percentage of the qualification

Assessment objective	Weighting in IGCSE %
AO1	40
AO2	40
AO3	20
Total	100

Assessment objectives as a percentage of each component

Assessment objective	Weighting in components %	
	Paper 1	Paper 2
AO1	60	20
AO2	20	60
AO3	20	20
Total	100	100

3 Subject content

This syllabus gives you the flexibility to design a course that will interest, challenge and engage your learners. Where appropriate, you are responsible for selecting resources and examples to support your learners' study. These should be appropriate for the learners' age, cultural background and learning context as well as complying with your school policies and local legal requirements.

Computer Science is a practical subject and a range of practical exercises must be integral to the teaching of this qualification. It is important that learners develop their computational thinking skills by doing practical problem-solving and programming using appropriate resources. It is also expected that learners have the opportunity in class to write their own programs in Python 3.10 or higher, as well as executing (running), testing and debugging them.

Any equipment and facilities should be adequate for learners to be able to satisfy the requirements of the syllabus. The hardware facilities needed will depend on the number of learners but must be sufficient for all learners to have enough time to practise their programming skills. Learners also need to have access to a system with direct-access file capability on backing store and hardcopy facilities.

Computer Systems and Logic

This section of content introduces the fundamental principles that underpin how computer systems work. Learners will explore how data is represented, processed and transmitted, from binary number systems and logic gates to networking and cybersecurity. Learners will also develop understanding of computer hardware, software, automated systems and emerging technologies such as artificial intelligence (AI) and quantum computing. Together, these topics provide a strong foundation for understanding modern computing systems and their impact on society, supporting learners to apply knowledge, analyse implications and evaluate the use of computing technologies in real-world contexts.

1 Data representation and logic gates

1.1 Number systems and logic gates

1.1.1 Know that computers internally represent all data in binary

1.1.2 Know that data is converted to binary, processed using logic gates

1.1.3 Identify and use the standard symbols for logic gates

1.1.4 Demonstrate the function of each logic gate limited to:

(a) NOT

(b) AND

(c) OR

(d) NAND

(e) NOR

(f) XOR

Logic gates will be limited to a maximum of two inputs

continued

1.1 Number systems and logic gates continued

1.1.5 Know how data storage is measured, limited to:
- (a) bit
- (b) nibble
- (c) byte
- (d) kibibyte (KiB)
- (e) mebibyte (MiB)
- (f) gibibyte (GiB)
- (g) tebibyte (TiB)

1.1.6 Convert between each data storage measurement in 1.1.5

1.1.7 Know the base of the denary, binary and hexadecimal number systems

1.1.8 Convert between:
- (a) denary and binary
- (b) denary and hexadecimal
- (c) hexadecimal and binary

Candidates will be expected to do conversions in both directions, values used will be positive integers only, maximum binary number length of 12 bits

1.1.9 Explain the advantages of using hexadecimal as an understandable representation of binary and identify examples where hexadecimal is used

1.1.10 Add two positive 8-bit binary integers

1.1.11 Describe overflow and how it can create errors in binary addition

1.1.12 Perform logical left and right shifts of multiple places on a positive 8-bit binary integer

1.1.13 Explain the effects of logical left and right shifts

1.1.14 Perform cyclic left and right shifts of multiple places on a positive 8-bit binary integer

1.1.15 Explain the effects of cyclic left and right shifts

1.1.16 Convert positive and negative binary or denary integers to their two's complement 8-bit representation

1.1.17 Convert two's complement 8-bit integers back to binary or denary

1.2 Text, sound and images

1.2.1 Know that computers store characters as binary numbers

1.2.2 Understand that a character set is a set of characters that have corresponding numeric codes

1.2.3 Describe how a computer represents text using character sets

1.2.4 Describe the features of the character sets:

- (a) American Standard Code for Information Interchange (ASCII)
- (b) Unicode

1.2.5 Describe the advantages and disadvantages of:

- (a) ASCII
- (b) Unicode

1.2.6 Describe how a computer represents analogue sound, including:

- (a) sample rate
- (b) sample resolution

1.2.7 Explain the effects of changing the sample rate and sample resolution

1.2.8 Know that a bitmap image is made up of pixels

1.2.9 Describe how a computer uses binary to represent a bitmap image, including:

- (a) resolution
- (b) colour depth

1.2.10 Explain the effects of changing the resolution or the colour depth on a bitmap image

1.3 Data compression

1.3.1 Explain the purpose of data compression for transmission and storage

1.3.2 Explain the impact of data compression for transmission and storage

1.3.3 Explain how images, sound and video files are compressed using lossy compression methods

1.3.4 Explain how text, images, video and sound files are compressed using the lossless compression methods of run length encoding (RLE) and Huffman coding

Candidates will **not** be required to create or interpret a Huffman tree

1.3.5 Describe the advantages and disadvantages of lossy and lossless compression methods and identify where each method would be appropriate from a given scenario

2 Logic circuits and hardware

2.1 Logic circuits

2.1.1 Use logic gates to create logic circuits from a:
- (a) problem statement
- (b) logic expression
- (c) truth table

Circuits must be created for the statement given, without simplification

Logic circuits will be limited to a maximum of three inputs and one output

2.1.2 Complete a truth table from a:
- (a) problem statement
- (b) logic expression
- (c) logic circuit

2.1.3 Write a logic expression from a:
- (a) problem statement
- (b) logic circuit
- (c) truth table

2.2 CPU architecture

2.2.1 Describe the role of the central processing unit (CPU) in a computer

2.2.2 Describe a microprocessor as a type of integrated circuit on a single chip

2.2.3 Describe the purpose of the components in a CPU, limited to the:
- (a) arithmetic logic unit (ALU)
- (b) control unit (CU)
- (c) program counter (PC)
- (d) memory address register (MAR)
- (e) memory data register (MDR)
- (f) current instruction register (CIR)
- (g) accumulator (ACC)
- (h) address bus
- (i) data bus
- (j) control bus
- (k) cache
- (l) cores
- (m) clock

2.2.4 Describe the roles of the central processing unit (CPU) and random access memory (RAM) in the von Neumann architecture

2.2.5 Describe the process of the fetch–decode–execute (FDE) cycle, including the role of each component in the CPU, in a computer that has a von Neumann architecture

2.2.6 Explain how the number of cores, size of the cache and the clock speed can affect the performance of a CPU

2.2.7 Describe the purpose of an instruction set for a CPU

2.3 Data storage

2.3.1 Explain the purpose and the role of primary storage in a computer, limited to:
- (a) random access memory (RAM)
- (b) read only memory (ROM)
- (c) cache
- (d) registers

2.3.2 Describe the similarities and differences between RAM and ROM

2.3.3 Explain the purpose and the role of secondary storage in a computer

2.3.4 Describe the similarities and differences between primary and secondary storage

2.3.5 Describe the operation of magnetic and solid-state storage, including how data is read from and written to each type of storage

2.3.6 Explain the differences between magnetic, and solid-state storage

2.3.7 Describe how virtual memory is created and used

2.3.8 Explain why virtual memory is necessary

3 Data transmission and networking

3.1 Packet switching

3.1.1 Understand that data is separated into packets to be transmitted

3.1.2 Describe the structure of a data packet, limited to:
- (a) the packet header
- (b) the payload
- (c) the trailer

3.1.3 State what is included in the packet header, the payload and the trailer

3.1.4 Describe the process of packet switching, including:
- (a) packets can take different routes
- (b) routers make forwarding decisions
- (c) at the destination packets are reassembled

3.2 Data transmission

3.2.1 Describe how data is transmitted from one device to another using different methods of data transmission, limited to:
- (a) serial
- (b) parallel
- (c) simplex
- (d) half-duplex
- (e) full duplex

3.2.2 Identify typical applications for each of the transmission methods in 3.2.1

3.2.3 Explain the suitability, advantages and disadvantages of the transmission methods in 3.2.1

3.2.4 Describe the purpose of a network interface card (NIC)

3.2.5 Describe the format and purpose of a media access control (MAC) address

3.2.6 State that each NIC has a unique identifier (MAC address)

3.2.7 Describe the function of a switch limited to storing the MAC addresses of connected devices

3.3 Internet Protocol Addresses

3.3.1 Describe the purpose of an internet protocol (IP) address in uniquely identifying networks and devices on the internet

3.3.2 Describe the format of an IP address

3.3.3 Explain the features of and the differences between IPv4 and IPv6 addressing, including:

- (a) IPv4 uses 32-bit addresses (written as four sets of denary numbers, separated by dots)
- (b) IPv6 uses 128-bit addresses (written as eight sets of hexadecimal numbers, separated by colons)

3.3.4 Explain the features of and the differences between static and dynamic IP addresses, including:

- (a) static IP addresses remain fixed and are often used for servers or hosting services
- (b) dynamic IP addresses are allocated temporarily by an ISP or home router when a device connects to the internet

3.3.5 Describe the functions of a router

3.4 The Internet and the World Wide Web

3.4.1 Describe the purpose of a uniform resource locator (URL)

3.4.2 Describe the structure of a URL

3.4.3 Describe the purpose and operation of hypertext transfer protocol (HTTP) as the protocol for requesting and delivering web resources

3.4.4 Describe the purpose and operation of hypertext transfer protocol secure (HTTPS) as a secure version of HTTP that uses encryption to ensure secure communication

3.4.5 Describe how web pages are located, retrieved and displayed on a device when a user inputs a URL, including the role of:

- (a) the web browser
- (b) IP addresses
- (c) domain name system (DNS)
- (d) the webserver
- (e) hypertext markup language (HTML)

3.4.6 Understand the role of the DNS in translating human-readable domain names into IP addresses so that browsers can locate web servers to make HTTP requests

3.4.7 Understand that cookies are small text files that store pieces of data stored by a web browser

3.4.8 Explain the difference between session cookies and persistent cookies

3.4.9 Describe the features of the cloud and cloud services

3.4.10 Explain the advantages and disadvantages of storing data on the cloud in comparison to storing data locally

3.4.11 Explain the difference between cloud storage models and when they would be used, limited to:

- (a) public
- (b) private
- (c) hybrid

4 Data integrity and cybersecurity

4.1 Data integrity

4.1.1 Describe how errors can occur during data storage, data transmission and data entry

4.1.2 Describe how methods detect errors and how errors are corrected, including:

(a) parity bit (odd and even)

(b) parity block check (odd and even)

(c) echo check

(d) checksum

(e) check digit

4.1.3 Calculate a parity bit or parity byte for a given piece of binary data

4.1.4 Identify examples of when a check digit is used, including:

(a) International Standard Book Number (ISBN)

(b) bar codes

(c) airline tickets

(d) bank account numbers

4.1.5 Describe how an automatic repeat reQuest (ARQ) can be used to acknowledge whether data is received without error, including the use of positive and negative acknowledgements

Candidates will **not** be required to calculate a check digit or checksum

4.2 Cybersecurity

4.2.1 State the core principles of cybersecurity as:

(a) confidentiality

(b) integrity

(c) authenticity

(d) availability

(e) non-repudiation

4.2.2 Describe how each of the core principles of cybersecurity can be threatened

4.2.3 Describe the aims and processes involved in cybersecurity threats, limited to:

(a) brute-force attack

(b) data interception

(c) distributed denial of service (DDoS) attack

(d) hacking and unauthorised access

(e) malware (virus, worm, Trojan horse, spyware, adware, ransomware)

(f) pharming

(g) phishing

(h) social engineering

(i) structured query language (SQL) injection

continued

4.2 Cybersecurity continued

4.2.4 Explain how solutions are used to help keep data safe from cybersecurity threats, limited to:
- (a) access levels
- (b) anti-malware, including anti-virus and anti-spyware
- (c) authentication (username and password, biometrics, two-step verification)
- (d) automating software updates
- (e) checking the spelling and tone of communications
- (f) checking the URL attached to a link
- (g) firewalls
- (h) privacy settings
- (i) proxy servers
- (j) transport layer security (TLS)
- (k) virtual private network (VPN)

4.2.5 Describe the purpose of encryption when storing or transmitting data, including:
- (a) protecting confidentiality
- (b) ensuring integrity
- (c) supporting authenticity and non-repudiation

4.2.6 Explain how data is encrypted using symmetric and asymmetric encryption methods

5 Software

5.1 Types of software and interrupts

5.1.1 Describe the purpose of an operating system as the software layer that manages the system resources that provides a platform for running application software and an interface for users to interact with the computer hardware

5.1.2 Describe how an operating system manages user accounts and system security

5.1.3 Describe how an operating system manages files, including:
- (a) file systems
- (b) directories

5.1.4 Describe how an operating system manages memory, including:
- (a) allocating memory to processes
- (b) loading processes into RAM
- (c) paging and segmentation
- (d) virtual memory
- (e) memory protection

5.1.5 Describe how an operating system manages processes, including:
- (a) scheduling processes
- (b) enabling inter-process communication

5.1.6 Describe how an operating system manages peripherals and device drivers

5.1.7 Describe how an operating system handles interrupts, including:
- (a) the conditions that cause hardware and software interrupts
- (b) how an interrupt is handled with an interrupt service routine (ISR)
- (c) how the operating system ensures the interrupted process can later resume

5.2 Programming languages, translators and IDEs

5.2.1 Describe the features of:
- (a) high-level languages
- (b) low-level languages, including assembly language
- (c) syntax

5.2.2 Explain the advantages and disadvantages of high-level and low-level languages

5.2.3 Describe the features and operation of:
- (a) a compiler
- (b) an interpreter
- (c) an assembler

5.2.4 Explain the advantages and disadvantages of each translator and identify the most appropriate for use in a given scenario

5.2.5 Describe the purpose of an integrated development environment (IDE) as a tool to support software development with various functionality provided, including:
- (a) code editors
- (b) run-time environment
- (c) translators
- (d) error diagnostics
- (e) auto-completion / code completion
- (f) prettyprint
- (g) AI assistance

6 Automated systems and robotics

6.1 Automated systems

6.1.1 Understand that a microcontroller is a complete computer system on a single chip, containing a processor, a small amount of RAM and ROM, and I/O ports

6.1.2 Understand that sensors are used to detect changes in the physical environment and an analog-to-digital converter (ADC) is used to convert analogue signals into digital data

6.1.3 Identify a suitable sensor for a given scenario

6.1.4 Describe the purpose and characteristics of an embedded system

6.1.5 Identify systems, devices and machines in which embedded systems are commonly used

6.1.6 Understand that actuators are devices that do actions in response to the output from a microcontroller

6.1.7 Describe how sensors, microcontrollers and actuators can be used in collaboration to create automated systems

6.1.8 Evaluate the advantages and disadvantages of using automated systems for:
- (a) transport
- (b) agriculture
- (c) domestic settings
- (d) retail

6.1.9 Evaluate the implications automated systems have on:
- (a) employment
- (b) safety
- (c) environmental sustainability

6.2 Robotics

6.2.1 Define robotics as a branch of technology that incorporates the design, construction, operation and use of robots

6.2.2 Describe the characteristics of a robot, including:

- (a) a mechanical structure or framework
- (b) electronic components, such as sensors, microcontrollers and networking hardware, motors, actuators and power supplies
- (c) programmability

6.2.3 Describe the roles that robots can perform in the following areas:

- (a) industry
- (b) agriculture
- (c) medicine

6.2.4 Evaluate the advantages and disadvantages of the use of robots in the areas listed in 6.2.3

6.2.5 Evaluate the implications of using robots for:

- (a) employment
- (b) safety
- (c) environmental sustainability

7 Emerging technologies

7.1 Artificial Intelligence

7.1.1 Understand that artificial intelligence (AI) is a branch of computer science focused on the simulation of intelligent behaviours by computers

7.1.2 Describe the main features of AI, including:

(a) the collection of data

(b) the rules for using that data

(c) the ability to process large data sets

(d) the ability to reason

(e) the ability to learn and adapt using supervised learning, unsupervised learning and reinforcement learning

7.1.3 Describe common applications of AI, limited to:

(a) recommendation systems

(b) natural language processing and large language models (LLMs)

(c) generative AI, including, images and video

(d) computer vision, including links to robotics and automated systems

(e) predictive analytics

7.1.4 Understand the structure of neural networks as the:

(a) input layer

(b) hidden layer(s)

(c) output layer

7.1.5 Explain the purpose of machine learning including how neural networks are used

7.1.6 Identify different ways that AI can be used in a given context

7.1.7 Evaluate the fairness and bias in data and AI systems

7.1.8 Evaluate the ethical and societal considerations of AI

7.1.9 Evaluate the implications of AI for:

(a) employment

(b) safety

(c) environmental sustainability

7.2 Quantum computers

7.2.1 Describe the differences between a quantum computer and a classical computer using terms such as:

(a) bit

(b) qubit

(c) electrons / photons

(d) binary

(e) logic gates

7.2.2 State the advantages and disadvantages of using quantum computers instead of classical computers

Candidates will **not** be required to explain quantum theory related to the function of a quantum computer

Algorithms and Programming

This section of content develops learners' ability to design, construct, test and evaluate algorithms and programs using Python. Learners will develop their understanding of programming constructs, data types, operators and structured programming techniques, as well as their ability to design and trace algorithms, identify and correct errors, and apply searching and sorting methods. Learners are also required to work with files, lists and relational databases, using SQL to retrieve and manipulate data, and to demonstrate effective testing and maintenance practices in response to given scenarios.

8 Programming fundamentals

8.1 Programming concepts

8.1.1 Understand the purpose of variables and constants and initialise them in Python

8.1.2 Identify and use the basic data types:

(a) (int) integer

(b) (float) real

(c) (str) string

(d) (bool) Boolean

8.1.3 Use casting to convert between the data types in 8.1.2

8.1.4 Identify and use inputs and outputs

8.1.5 Demonstrate and use the following programming constructs:

(a) sequence

(b) selection

- if, elif and else statements
- nested selection
- match-case

(c) iteration

- for loop (count-controlled)
- while loop (pre-condition)
- while, break (post-condition loop)
- nested iteration (loops)

(d) Understand and use the concepts of totalling and counting

8.1.6 Demonstrate and use the concept of string handling, including:

(a) `len()` length

(b) `.find()` find the position of a substring

(c) `.count()` count substrings

(d) `.upper()` uppercase

(e) `.lower()` lowercase

(f) `.title()` title case

(g) `.replace()` replace

(h) `+` concatenation

The first character of the string is position zero

8.2 Programming operators

8.2.1 Demonstrate and use arithmetic operators, limited to:
- (a) + (addition)
- (b) – (subtraction)
- (c) / (float division)
- (d) * (multiplication)
- (e) ** (raise to the power of)
- (f) % (modulus division)
- (g) // (floor division)

8.2.2 Demonstrate and use relational operators, limited to:
- (a) == equal to
- (b) != not equal to
- (c) < less than
- (d) <= less than or equal to
- (e) > greater than
- (f) >= greater than or equal to

8.2.3 Demonstrate and use logical operators, limited to:
- (a) and
- (b) or
- (c) not

9 Algorithm design

9.1 Abstraction

9.1.1 Describe abstraction as the process of creating a simplified model that represents the essential features of a problem

9.1.2 Explain the purpose of abstraction for a given context

9.1.3 Demonstrate how to use abstraction

9.2 Decomposition

9.2.1 Describe decomposition as the process of breaking down problems into smaller, more manageable sub problems

9.2.2 Describe how a problem can be decomposed into its component parts, limited to:
- (a) inputs
- (b) processes
- (c) outputs
- (d) storage

9.2.3 Explain the purpose of decomposition for a given context

9.2.4 Demonstrate how to use decomposition

9.3 Algorithm design and evaluation

9.3.1 Design, complete and amend algorithms for a given context using:
- (a) a flowchart
- (b) Python code

9.3.2 Explain the purpose of a given algorithm and the purpose of the component parts, limited to:
- (a) inputs
- (b) outputs
- (c) processes
- (d) storage

9.3.3 Complete a trace table for a given algorithm including inputs, outputs and variables for a given set of data

9.3.4 Identify and correct errors in a given algorithm

9.3.5 Amend an algorithm

9.3.6 Show how searching algorithms are performed on a given set of data, limited to:
- (a) linear search
- (b) binary search

9.3.7 Show how sorting algorithms are performed on a given set of data, limited to:
- (a) bubble sort
- (b) insertion sort
- (c) merge sort

9.3.8 Construct the algorithms using a flowchart or Python for a:
- (a) linear search
- (b) bubble sort

9.3.9 Describe the steps involved in a:
- (a) linear search
- (b) binary search
- (c) bubble sort
- (d) merge sort

9.3.10 Compare and contrast the features and efficiency of algorithms, limited to:
- (a) linear search
- (b) binary search
- (c) bubble sort
- (d) merge sort

Candidates will **not** be expected to use Big-O notation when comparing algorithms

Flowcharts may contain multiple statements in one box

A decision box may combine a process and a decision

10 Further programming and testing

10.1 Variables and constants

10.1.1 Demonstrate and use local and global variables

10.1.2 Demonstrate and use local and global constants

10.1.3 Explain the advantages and disadvantages of local and global:

(a) variables

(b) constants

10.2 Procedures and functions

10.2.1 Identify procedures and functions in a given context

10.2.2 Explain the purpose and concept of procedures and functions in a given context

10.2.3 Demonstrate and use functions, including:

(a) `sum()`	totalling
(b) `.count()`	counting
(c) `max()`	maximum
(d) `min()`	minimum
(e) `.mean()`	mean average
(f) `round()`	rounds a number
(g) `random.randrange()`	random number generation
(h) `random.randint()`	random number generation

10.2.4 Demonstrate and use procedures and call functions with or without parameters

10.3 Lists

10.3.1 Explain the purpose of a list including its features

10.3.2 Define, initialise and use one-dimensional (1D) and two-dimensional (2D) lists, including:

(a) a variable for the index

(b) writing into, and reading from a list including the use of iteration

(c) iterating through each element in a list

The first index is zero

10.3.3 Use Python's built-in functions and operators for working with lists, including:

(a) `sum()`

(b) `max()`

(c) `min()`

(d) `len()`

(e) `.append()`

(f) `.insert()`

(g) `.pop()`

(h) `.sort()`

(i) `in, not in` (membership)

continued

10.3 Lists continued

10.3.4 Write algorithms for a:

(a) linear search

(b) bubble sort

10.4 File handling

10.4.1 Explain the purpose of storing data in a file to be used by a program

10.4.2 Demonstrate and use:

(a)	`open()`	open a file
(b)	`.close()`	close the file
(c)	`with open()`	
(d)	`"r"`	open in read mode
(e)	`"w"`	open in write mode
(f)	`"a"`	open in append mode
(g)	`.read()`, `.readline()`, `.readlines()`	read the file
(h)	`.write()`	write to file

10.5 Testing

10.5.1 Describe the purpose of validation checks, limited to a:

(a) range check

(b) length check

(c) type check

(d) presence check

(e) format check

(f) check digit

10.5.2 Write Python code to validate input data for a given scenario, (inclusive and exclusive)

10.5.3 Explain the purpose of verification, including the function of visual and double entry checks

10.5.4 Write Python code to do a double entry check

10.5.5 Suggest and apply suitable normal, boundary and erroneous test data

10.5.6 Identify and debug errors in given Python code

10.5.7 Identify and explain the different types of programming errors, limited to:

(a) syntax error

(b) logic/logical errors

(c) run-time errors

10.5.8 Demonstrate and use techniques that make a program easier to maintain, including:

(a) meaningful identifiers

(b) appropriate commenting

(c) procedures and functions

(d) relevant program layout

11 Databases and Structured Query Language (SQL)

11.1 Databases

11.1.1 Define a single-table database from given data storage requirements, including:
- (a) fields
- (b) records
- (c) validations

11.1.2 Define a two-table relational database from given data storage requirements, including:
- (a) fields
- (b) records
- (c) validations

11.1.3 Identify data types, limited to:
- (a) text
- (b) character
- (c) Boolean
- (d) integer
- (e) decimal
- (f) date
- (g) time
- (h) currency

11.1.4 Explain the purpose of a primary key

11.1.5 Identify a suitable primary key for a given table

11.1.6 Explain the purpose of a foreign key to establish a link and join two tables

11.1.7 Identify a suitable foreign key for a given database

11.1.8 Demonstrate how two tables can be joined in a database

11.2 SQL

11.2.1 Write structured query language (SQL) statements using one or more conditions to retrieve data from a one and two table database

11.2.2 Correct SQL statements

11.2.3 Explain and demonstrate the result for a given SQL statement, limited to:
- (a) `SELECT, FROM, WHERE`
- (b) `INSERT INTO`
- (c) `ORDER BY DESCENDING`
- (d) `ORDER BY ASCENDING`
- (e) `SUM`
- (f) `COUNT`
- (g) `AND`
- (h) `OR`
- (i) `INNER JOIN, ON`

4 Details of the assessment

Paper 1 – Computer Systems and Logic

Written paper, 1 hour 45 minutes, 75 marks

This question paper consists of short-answer and structured questions set on Topics 1–8 of the subject content.

All questions are compulsory, and candidates answer on the question paper.

This paper assesses all assessment objectives, AO1, AO2 and AO3, and assesses the full grade range, 9 to 1.

This paper is externally assessed.

Calculators are **not** allowed in this examination.

Paper 2 – Algorithms and Programming

Written paper, 1 hour 45 minutes, 75 marks

This question paper consists of short-answer and structured questions set on Topics 8–11 of the subject content.

All questions are compulsory, and candidates answer on the question paper.

Questions require candidates to have practical programming experience of Python 3.10 or higher and they will be required to design and write programs or solve problems using Python 3.10 or higher.

This paper assesses all assessment objectives, AO1, AO2 and AO3, and assesses the full grade range, 9 to 1.

This paper is externally assessed.

Calculators are allowed in this examination.

Scenario

The final question in Paper 2 presents an unseen scenario where candidates will be required to write code in Python 3.10 or higher for the context provided.

The scenario uses the methods and concepts listed in Topics 8–11 of the subject content.

The scenario will consist of multiple parts with the total mark for the whole question being 15 marks.

It is expected that candidates should spend 30 minutes answering the questions related to the scenario.

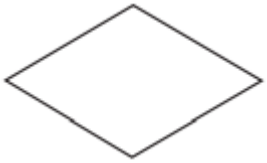
Teachers are advised to familiarise themselves with the updated Paper 2 specimen paper and mark scheme for first assessment 2029.

Mathematical requirements

Candidates should be able to:

- add, subtract, multiply and divide
- use averages, random numbers, decimals, fractions, percentages and ratios
- use both positive and negative integers, and real numbers
- use arithmetic and logical operators
- use different number systems, including binary, denary and hexadecimal
- use methods of counting, totalling and rounding.

Flowchart symbols

	**Flow line**	An arrow represents control passing between the connected shapes.
	**Process**	This shape represents something being performed or done.
	**Subroutine**	This shape represents a subroutine call that will relate to a separate, non-linked flowchart.
	**Input/Output**	This shape represents the input or output of something into or out of the flowchart.
	**Decision**	This shape represents a decision (Yes/No or True/False) that results in two lines representing the different possible outcomes.
	**Terminator**	This shape represents the ‘Start’ and ‘Stop’ of the process.

Logic gate symbols

	NOT
	AND
	OR
	NAND
	NOR
	XOR (EOR)

Python 3.10 guide

The following information sets out how the Python programming language will appear within the examinations of this syllabus.

General style

Font style and size

Python code is presented in Courier New. The size of the font will be consistent throughout.

Indentation

Lines are indented by four spaces to indicate that they are contained within a statement in a previous line.

1.1 Identifiers

When naming variables, procedures and functions, the following rules must be observed:

- names must begin with character 'a'–'z' or 'A'–'Z' or '_' and followed by alphanumeric characters or '_'
- names cannot contain spaces
- reserved words cannot be used
- identifiers are case sensitive
- constants are identified by being named in all capital letters

Indentation is a fundamental syntax aspect of Python. It defines the beginning and end of code blocks including functions, procedures, selection and iteration. In nested statements the indentation level indicates which statements belong to each block and the order in which they are executed.

1.2 Assignment

Statement	Notes
`score = 1`	normal assignment
`score += 1`	augmented assignment equivalent to score = score + 1

1.3 Data types

Type	Example	Notes
`int`	`45`	integer, whole number which cannot have a decimal place
`float`	`3.1415926`	floating point number which will have a decimal place
`bool`	`True` `False`	Boolean, one of the two values which must start with a capital letter
`str`	`"Hello"` `'Hello'`	string, alphanumeric values contained inside `""` or `''` string values can be enclosed in either single quotes `'` or double quotes `"`. In this guide, double quotes `"` are used for consistency.

1.4 Inputs and outputs

```
name = input("what is your name?")
print("Accepted")
```

1.5 Casting

Type	Example	Notes

`str(data)`	`str(1.23)`
	`str(price)`
	`price = str(price)`
		converts data to type string

`int(data)`	`int("11")`
	`int(age)`
	`age = int(age)`
		converts data to type integer

`float(data)`	`float("1.23")`
	`float(length)`
	`length = float(length)`
		converts data to type float

Casting can also be done on input, e.g. `age = int(input("How old are you?"))`

1.6 Comments

Comments are preceded by a hash mark: #. The comment continues until the end of the line.

Normally the comment is on a separate line before, and at the same level of indentation as, the code it refers to. Occasionally, however, a short comment that refers to a single line may be at the end of the line to which it refers.

```
# This is a comment
```

1.7 Arithmetic operators

Operator	Use
+	addition
–	subtraction
/	float division
*	multiplication
**	raise to the power of
%	modulus division
//	floor division

1.8 Relational operators

Operator	Use
`==`	equal to
`!=`	not equal to
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to

1.9 Logical operators

Logical Expression	Use
X `and` Y	logical and
X `or` Y	logical or
`not` X	logical not

1.10 String handling

The first character of the string is index zero

Operator	Use
`+`	concatenate

Example

```
full_name = first_name + " " + last_name
```

Concatenates data of type string without adding additional spaces.

```
print("Hello " + name)
```

Concatenation can be combined with other statements.

Operator	Use
`len()`	length

Example

```
name_length = len(first_name)
```

Returns the number of characters in the string as an integer.

continued

1.10 String handling continued

Operator	Use
`.find()`	find a substring

Example

```
location = text.find("computer")
```

Returns the index of the first instance of the string `"computer"` in the variable `text` and stores the result of the calculation in the variable `location`

Operator	Use
`.count()`	count substrings

Example

```
vowel_a = word.count("a"))
```

Returns how many times `"a"` appears in the variable `word` and stores the result in the variable `vowel_a`

Operator	Use
`.upper()`	uppercase

Example

```
loud_version = greeting.upper()
```

Returns a modified copy of the string `greeting` with all characters in uppercase and stores it in the variable `loud_version`

Operator	Use
`.lower()`	lowercase

Example

```
address = address.lower()
```

Stores the contents of `address` in lower case.

Operator	Use
`.title()`	title case

Example

```
print(greeting.title())
```

Outputs the contents of the variable `greeting` in title case.

continued

1.10 String handling continued

Operator	Use
`.replace()`	replace

Example

```
address = address.replace("a","A")
```

Stores contents of `address` with all the values `"a"` replaced with the values `"A"`.

Operator	Use
`X[0]`	indexing

Example

```
initial = name[0]
```

Returns the first character of `name` and stores it in the variable `initial`

```
ending = name[-1]
```

Returns the last character of `name` and stores it in the variable `ending`

Operator	Use
`X[0:0]`	string slicing range

Example

```
phrase = sentence[5:15]
```

Returns the characters from index 5, up to but not including index 15 of the variable `sentence` and stores it in the variable `phrase`

1.11 Selection

1.11.1 If statements

Levels of selection	Use
`if condition(s):` `    <statement(s)>`	selection that checks one condition

Example

```
if passcode == 4323:
   print("passcode accepted")
```

The variable `passcode` is checked to see if it is equal to or has the value of `4323` and if so the print statement is output.

Levels of selection	Use

`if condition(s):`
`    <statement(s)>`
`else:`
`    <statement(s)>`
	selection that checks one condition, it not met executes the statements after the else clause

Example

```
if age <5:
    price = 0
else:
    price = 3
```

The variable `age` is checked to see if it is a value less than `5` and assigns the variable `price` a value of `0` if the condition is `True` and a value of `3` if the condition is not true.

continued

1.11.1 If statements continued

Levels of selection	Use

`if condition(s):`
`    <statement(s)>`
`elif condition(s):`
`    <statement(s)>`
`else:`
`    <statement(s)>`
	selection that checks multiple conditions in order executing just one of them. Can be with or without an else clause

Example

```
if adults >=2 and children >=2:
    print("family ticket is cheaper")
elif adults >8:
    print("group ticket is cheaper")
else:
    print("cheapest tickets selected")
```

The variable `adults` is checked to see if it hsa the value of is equal to a number of 2 or more, and the variable `children` is checked to see if it contains a number of 2 or more, if so the print statement on the next line is output and the if statement ends.

If the previous clause evaluates to false the next clause is checked to see if the number of adults stored in the variable `adults` is greater than 8, if so the print statement on the next line is output and the if statement ends.

If neither of the previous clauses were true the else clause is executed and the last print statement is output.

Levels of selection	Use

`if condition(s):`
`    if condition(s):`
`        <statement(s)>`
	nested if statements can have multiple levels and may or may not have an else case for each statement

Example

```
if joined == True
    if passcode == 1323:
        print(“passcode accepted”)
```

The variable `joined` is check to see if it contains the value `True`, then the variable `passcode` is checked to see if it contains the data `1323` and if so the message in the print statement is output.

1.11.2 Match case

Match case statement	Notes

`match x:`
`  case value(s):`
`  <statement(s)>`
`  case _:`
`  <statement(s)>`
	for selecting individual values
	`case _` may or may not be used, it is for any other scenario that is not already met

Example

```
match price:
    case 0:
        print("already paid")
    case _:
        print(price)
```

The variable `price` is checked to see if it contains the value `0` and if so the data in the print statement `print("already paid")` is output and the match case statement ends.

If the value in the variable `price` is anything other than `0` then the data stored in the variable `price` is output.

Match case statement	Notes

`match x:`
`  case 0 | 1 | 2:`
`  <statement(s)>`
	for selecting one or more values in one clause

Example

```
match ticket:
    case "elderly" | "student":
       price = price * 0.5
```

The variable `ticket` is checked to see if it contains the data `"elderly"` or `"student"`, if it does then the `price` is multiplied by 0.5 and stored in the variable `price`

1.12 Lists and 2D lists

List indexes start at 0

Initialising a list:

```
pupils = [] # this is an empty list

animals = ["camel", "lion", "sheep"]

entries = [1, 3, 5, 6, 2]
```

Elements of a list can be accessed using their index:

```
animals[0]  # would return the data from the index 0 of the list animals
```

Lists indexes can be assigned directly using their index number. In the example a new value of 3 is assigned to the element at the specified index 0 in the list, overwriting the value previously stored at that index.

```
entries[0] = 3
```

Initialising a 2D list:

```
zoo_animals =[["camel", 4],["lion", 3]]
```

1.13 List functions

Function	Notes
`sum()`	returns the sum of all the values in a list

Example

```
total = sum(entries)
```

Adds up all the values of the list `entries` and stores it in the variable `total`

Function	Notes
`max()`	returns the highest value in a list

Example

```
winning_score = max(scores)
```

Returns the highest value from the list `scores` and stores it in the variable `winning_score`

Function	Notes
`min()`	returns the lowest value in a list

Example

```
lowest_temp = min(temperatures)
```

Returns the lowest value from the list `temperatures` and stores it in the variable `lowest_temp`

continued

1.13 List functions continued

Function	Notes
`.count()`	counts how many occurrences of the given value are in the list

Example

```
couples = bookings.count(2)
```

Counts the number of times the data `2` is in the `bookings` list and stores it in the variable `couples`

Function	Notes
`len()`	returns the number of items in the list

Example

```
if len(animals) <4:
```

This selection statement checks if there are less than 4 animals stored in the list.

Function	Notes
`.append()`	extends the list by one element aading the item to the end

Example

```
if len(animals) <4:
    animal = input("enter another animal")
    animals.append(animal)
```

This selection statement allows an additional animal to be inputted into the variable `animal` and appended to the list `animals` if there are less than 4 items of data stored in the list.

Function	Notes
`insert`	takes two parameters, index and value. Inserts the value into index of the list. Extending the list and moving following elements up 1 index value.

Example

```
animals.insert(1, "chicken")
```

This extends the list `animals` by one index, the values in index 1 and above are moved up one place. Then it adds the value `"chicken"` into the index 1 of the list `animals`

continued

1.13 List functions continued

Function	Notes
`pop`	function requires one parameter which is the index of the data to be removed. It returns and removes the value at the given index of the list. If no index is given then it defaults to the last item in the list

Example

```
animals.pop()
```

This would remove and return the last item from the list `animals`

Function	Notes
`sort`	sorts list into order. Defaults to ascending order unless supplied with the argument reverse = True.

Example

```
animals.sort()
```

Sorts the list animals into ascending order.

```
animals.sort(reverse=True)
```

Sorts the list animals into descending order.

Function	Notes
`in` `not in`	Used for membership, it checks if a value is in or is not in a list and returns a Boolean result

Example

```
if "pink" in colours:
    print("pink is not available")
if blue not in colours:
    print("blue is still available")
```

The list `colours` is checked to see if the value `"pink"` and `"blue"` have already been used and a corresponding message is displayed.

1.14 Iteration

Construct	Notes
`while condition(s):` `    <statement(s)>`	pre-condition loop iterates while the condition is true

Example

```
while passcode !=323:
    print("invalid, input again")
    passcode = int(input())
```

Loop condition checks if the value stored in `passcode` is not equal to `323` and executes the statements.

The loop will end when the passcode is input as `323`.

Construct	Notes

`while condition(s):`
`    <selection statement>:`
`        break`
	post condition loop
	Selection is used to check the condition and end the loop

Example

```
while True:
    if continue =="no":
        break
    item = input("next item?")
    items.append(item)
    continue = input("more items?")
```

The selection statement checks to see if the value stored in the variable `continue` is equal to `"no"`. If it is then the loop ends with the statement `break`, otherwise the rest of the code block is run.

continued

1.14 Iteration continued

Construct	Notes
`for <counter> in range:` `    <statement(s)>`	count controlled loop the counter is initialised and incremented by the looping construct for a preset number of iterations

Example

```
for counter in range(10):
    print(animals[counter])
```

Outputs the data stored in indexes 0 to 9 of the list animals.

```
for counter in range(5, 10):
    print(animals[counter])
```

Outputs the data stored in indexes 5 to 9 of the list animals.

```
total = 0
for count in range(5, 100, 2):
     total = total +count
```

`total` is initialised at 0. Loop counter `count` will be initialised as `5` and be increased from `5` to `99`, increasing the counter by `2` on each iteration instead of the default of `1`

Construct	Notes
`for <element> in list:` `    <statement(s)>`	count controlled loop loop iterates for each index of the list counter stores the data that is contained in each that indexes location

Example

```
for student in students:
    print(student + "enter score")
    score = int(input())
    scores.append(score)
```

Student names are stored in the list `students`. Student scores are stored in the list `scores`. The index starts at 0 and increases by one for each iteration. The variable `student` stores the data at the current index so that the student name can be output and their score requested to be appended to the list `scores`

1.15 Procedures and functions

1.15.1 Defining and calling

Procedures and functions are declared as:

```
def <function name> (parameters):
```

functions return a value using:

```
return <value>
```

A function can be designed to accept zero, one, or many arguments. When called, it must be supplied with the exact number of arguments its definition requires.

```
function_name(argument1, argument2, ...)
```

Example

```
def add_tax(price,percentage):  # function definition
    new_price = price *(1+(percentage/100)) #updated value with tax
    return new_price #returning the updated value

with_tax = add_tax(500, 10)
#function called, result stored in with_tax
```

1.15.2 Built-in functions

Function	Notes
`import statistics` `statistics.mean()`	returns the mean average of a data set data set can be list or sequence of data

Example

```
import statistics
data = [10, 12, 23, 15, 16]
average = statistics.mean(data)
```

Returns the average of the values stored in the list `entries` and stores it in the variable `average`

Function	Notes
`round()`	returns a floating point number rounded to the given number of decimal places, if no decimal places passed it returns the number as an integer

Example

```
area = round(area,2)
```

Rounds the value stored in `area` to two decimal places and stores the returned result in the variable `area`

```
area = round(area)
```

Rounds the value stored in `area` to the nearest integer and stores the returned result in the variable `area`

continued

1.15.2 Built-in functions continued

Function	Notes
`import random` `random.randrange(0, 0)`	returns a random integer between the start and end values, not including the end value

Example

import random
`number = random.randrange(1, 50)`

Generates a random number between 1 and 49 which when returned is stored in the variable `number`

Function	Notes
`import random` `random.randint(0, 0)`	returns a random integer between the start and end values, including the end value

Example

import random
`number = random.randint(1, 50)`

Generates a random number between 1 and 50 which when returned is stored in the variable `number`

1.16 Local and global variables and constants

Variables and constants declared outside of procedures and functions are global by default and can be used anywhere in the program.

Variables and constants declared inside procedures and functions are local to that specific procedure or function (and their values can only be accessed within the procedure or function).

To assign a global variable inside a procedure or function, use `global` before assignment.

```
def costings():

    global year

    year = 1997
```

This will update the value of `year` in the main program and not create a local copy within the procedure `costings`.

1.17 File handling

1.17.1 Mode of opening files

Mode	Notes
write	opening a file in write mode deletes the existing file contents before writing new data the read methods cannot be used when a file is opened in write mode.
read	opening a file in read mode, allowing for the use of the read methods the file cannot be written to when opened in read mode
append	opening a file in append mode keeps data that is already in the file data written to the file is appended at the end the read methods cannot be used when a file is opened in append mode

1.17.2 Methods for opening files

Method 1

The file is opened in the mode required and closed manually.

Example:

```
students = open("students.txt", "a")
<file operations>
students.close()
```

Method 2

The file is opened using with open, the file operations happen in the indented section of code and the file is then closed when the indented code has ended.

Example:

```
with open("students.txt", "a") as students:
    <file operations>
```

Using this method the file is not manually closed.

1.17.3 File operations

Operation	Notes
`read`	reads the whole file, returning a string.

Example

```
day1_scores = scores.read()
```

Stores the entire contents of the file opened into the and stores it in the variable `day1_scores`

Operation	Notes
`readline`	reads one line, returning a string

Example

```
player1 = players.readline()
player2 = players.readline()
```

Store the first line of the file opened into the variable `player1` and the second line of the file opened in `player2`

Operation	Notes
`readlines`	reads the whole file, returning a list of strings, where each element of the list is one line of the file

Example

```
all_players = players.readlines()
```

Each line of the file stored in players is stored as an element of the list `all_players`

Operation	Notes
`write`	writes the specified string value to the file

Example

```
high_score.write("150")
```

Writes the value `150` to the file opened into `high_score`

1.17.4 File handling examples

Technique	Use
`X = open("file.txt","a")`	opening a file in append mode keeps data that is already in the file
`X = open("file.txt","w")`	opening a file in write mode overwrites data that is already in the file
`X = open("file.txt","r")`	opening a file in read mode
`X.close()`	closes the file
`with open("file.txt" [JW2.1][JH2.2] as X):` `    <file operations>`	uses indentation to define file operations and automatically closes the file when the block ends
`X.read()`	reads the whole file
`X.readline()`	reads one line
`X.readlines()`	reads the whole file as a list with each line being a new element in the list
`X.write(Y)`	writes the data Y to the file X

Command words

Command words and their meanings help candidates know what is expected from them in the exams. The table below includes command words used in the assessment for this syllabus. The use of the command word will relate to the subject context.

Command word	What it means
Calculate	work out from given facts, figures or information
Compare	identify/comment on similarities and/or differences
Convert	to change a number from one number system (base) to another
Define	give precise meaning
Describe	state the points of a topic / give characteristics and main features
Evaluate	judge or calculate the quality, importance, amount, or value of something
Explain	set out purposes or reasons / make the relationships between things clear / say why and/or how and support with relevant evidence
Give	produce an answer from a given source or recall/memory
Identify	name/select/recognise
Outline	set out the main points
Show (that)	provide structured evidence that leads to a given result
State	express in clear terms
Suggest	apply knowledge and understanding to situations where there are a range of valid responses in order to make proposals / put forward considerations

5 What else you need to know

This section is an overview of other information you need to know about this syllabus. It will help to share the administrative information with your exams officer so they know when you will need their support. Find more information about our administrative processes at **www.cambridgeinternational.org/eoguide**

Before you start

Previous study

We recommend that learners starting this course should have studied a broad curriculum such as the Cambridge Lower Secondary programme or equivalent national educational framework.

Guided learning hours

We design Cambridge IGCSE syllabuses to require about 130 guided learning hours for each subject. This is for guidance only. The number of hours a learner needs to achieve the qualification may vary according to each school and the learners' previous experience of the subject.

Availability and timetables

All Cambridge schools are allocated to one of six administrative zones. Each zone has a specific timetable. Find your administrative zone at **www.cambridgeinternational.org/adminzone**

You can view the timetable for your administrative zone at **www.cambridgeinternational.org/timetables**

You can enter candidates in the June and November exam series.

Check you are using the syllabus for the year the candidate is taking the exam.

Private candidates can enter for this syllabus. For more information, please refer to the *Cambridge Guide to Making Entries*.

Combining with other syllabuses

Candidates can take this syllabus alongside other Cambridge International syllabuses in a single exam series. The only exceptions are:

- Cambridge IGCSE Computer Science (0478)
- Cambridge IGCSE Computer Science (0265)
- Cambridge O Level Computer Science (2210)
- syllabuses with the same title at the same level.

Cambridge IGCSE, Cambridge IGCSE (9–1) and Cambridge O Level syllabuses are at the same level.

Group awards: Cambridge ICE

Cambridge ICE (International Certificate of Education) is a group award for Cambridge IGCSE. It encourages schools to offer a broad and balanced curriculum by recognising the achievements of learners who pass exams in a range of different subjects.

Learn more about Cambridge ICE at **www.cambridgeinternational.org/cambridgeice**

Making entries

Exams officers are responsible for submitting entries. We encourage them to work closely with you to make sure they enter the right number of candidates for the right combination of syllabus components. Entry option codes and instructions for submitting entries are in the *Cambridge Guide to Making Entries*. Your exams officer has access to this guide.

Exam administration

To keep our exams secure, we produce question papers for different areas of the world, known as administrative zones. We allocate all Cambridge schools to an administrative zone determined by their location. Each zone has a specific timetable.

Some of our syllabuses offer candidates different assessment options. An entry option code is used to identify the components the candidate will take relevant to the administrative zone and the available assessment options.

Support for exams officers

We know how important exams officers are to the successful running of exams. We provide them with the support they need to make entries on time. Your exams officer will find this support, and guidance for all other phases of the Cambridge Exams Cycle, at **www.cambridgeinternational.org/eoguide**

Retakes

Candidates can retake the whole qualification as many times as they want to.

Learn more about retake entries, including definitions and information on entry deadlines, at **www.cambridgeinternational.org/retakes**

To confirm what entry options are available for this syllabus, refer to the *Cambridge Guide to Making Entries* for the relevant series. Regulations for carrying forward component marks can be found in the *Cambridge Handbook* for the relevant year of assessment at **www.cambridgeinternational.org/eoguide**

Language

This syllabus and the related assessment materials are available in English only.

Accessibility and equality

Syllabus and assessment design

At Cambridge we recognise that our candidates have highly diverse linguistic, cultural and socio-economic backgrounds, and may also have a variety of protected characteristics. Protected characteristics include special educational needs and disability (SEND), religion and belief, and characteristics related to gender and identity.

We apply accessible design principles to make our syllabuses and assessment materials as accessible and inclusive as possible. This includes reviewing the accessibility of language, layout, contexts and visual content. Using this approach means that we give all candidates the fairest possible opportunity to demonstrate their knowledge, skills and understanding.

Access arrangements

Our design principles aim to make sure our assessment materials are accessible for all candidates. To further minimise barriers faced by candidates with SEND, illness or injury, we offer a range of access arrangements and modified papers. This is the principal way in which we comply with our duty to make 'reasonable adjustments', as guided by the UK Equality Act 2010.

Important:

Requested access arrangements should be based on evidence of the candidate's barrier to taking an assessment and should also reflect their normal way of working. For Cambridge to approve an access arrangement, we need to agree that it constitutes a reasonable adjustment and does not affect the security or integrity of the assessment. Our access arrangement regulations can be found in section 1.3 of the *Cambridge Handbook* **www.cambridgeinternational.org/eoguide**

Applying for access arrangements

- Details of our standard access arrangements and modified question papers are available in section 1.3 of the *Cambridge Handbook* **www.cambridgeinternational.org/eoguide**
- Centres are expected to check the availability of access arrangements and modified question papers at the start of the course. Check the *Cambridge Handbook*, the assessment objectives listed in the syllabus document and, where applicable, any access arrangement restrictions listed in the syllabus document.
- Contact us at the start of the course to find out if we can approve an access arrangement that is not listed in the *Cambridge Handbook*.
- All applications should be made by the deadlines published in the *Cambridge Handbook*.

After the exam

Grading and reporting

Grades 9, 8, 7, 6, 5, 4, 3, 2 or 1 indicate the standard a candidate achieved at Cambridge IGCSE (9–1).

9 is the highest and 1 is the lowest. 'Ungraded' means that the candidate's performance did not meet the standard required for grade 1. 'Ungraded' is reported on the statement of results but not on the certificate.

In specific circumstances, your candidates may see one of the following letters on their statement of results:

- Q (PENDING)
- X (NO RESULT).

These letters do not appear on the certificate.

On the statement of results, Cambridge IGCSE is shown as INTERNATIONAL GENERAL CERTIFICATE OF SECONDARY EDUCATION (IGCSE).

On certificates, Cambridge IGCSE is shown as International General Certificate of Secondary Education.

How students and teachers can use the grades

Assessment at Cambridge IGCSE has two purposes:

1 to measure learning and achievement

The assessment confirms achievement and performance in relation to the knowledge, understanding and skills specified in the syllabus.

2 to show likely future success

The outcomes help predict which students are well prepared for or likely to be successful in a particular course or career.

The outcomes help students choose the most suitable course or career.

Changes to this syllabus for 2029, 2030 and 2031

The syllabus has been reviewed and revised for first examination in 2029.

You must read the whole syllabus before planning your teaching programme.

Changes to version 1 of the syllabus, published September 2026

Changes to syllabus content	We have updated the syllabus content and guidance to reflect changes in content, scope and approach. Learning objectives have been written to be accessible and clear. Computer Systems and Logic In particular, we now require explicit understanding of: • IPv4 and IPv6 • Static and dynamic IP addressing • Session and persistent cookies • We have introduced cloud storage models, requiring candidates to understand public, private and hybrid cloud storage and their appropriate uses. • We have clarified and expanded the principles of cybersecurity, including confidentiality, integrity, authenticity, availability and non repudiation. • We have significantly expanded the artificial intelligence content, including: – types of machine learning – neural network structure – ethical issues, bias and fairness • We have introduced quantum computing at a conceptual level, requiring understanding of how it differs from classical computing. Algorithms and Programming • We now require all candidates to respond to programming questions using Python 3.10 or higher. • We have clarified expectations about methods and concepts required in Paper 2, ensuring alignment with topics 8 to 11. Command Words • We have added the command word **Convert** which means, 'to change a number from one number system (base) to another.' continued

Changes to syllabus content continued	**Content removed from the syllabus** We have removed and reduced the content. These topics are **no longer explicitly assessed.** Computer Systems and Logic We have removed the requirement for candidates to **calculate file sizes** for: • Bitmap images • Sound files We have removed data storage calculations using: • Pebibytes (PiB) • Exbibytes (EiB) We have removed explicit coverage of the **universal serial bus (USB)** interface. Legacy media devices and storage have been removed. We have limited automated systems to the following areas: • Transport • Agriculture • Domestic settings • Retail • Industry • Medicine We have removed the entire topic on **digital currency**. Algorithms and Programming We have removed explicit reference to the **program development life cycle (PDLC)** We have removed the requirement for candidates to use **pseudocode**, although an awareness of the purpose of pseudocode is still valuable.
Changes to assessment (including changes to specimen papers)	• All programming responses in paper 2 must be written using Python 3.10 or higher. • The scenario question has been adapted and will now consist of multiple separate sub parts to support accessibility. • The mark scheme for the scenario question is now points-based and **not** levels-based.

Significant changes to the syllabus are indicated by black vertical lines either side of the text.

In addition to reading the syllabus, you should refer to the updated specimen assessment materials. The specimen papers will help your students become familiar with exam requirements and command words in

questions. The specimen mark schemes show how students should answer questions to meet the assessment objectives.

Syllabuses and specimen materials represent the final authority on the content and structure of all of our assessments.

With a Customer Services team available 24 hours a day, 6 days a week, and dedicated regional teams supporting schools in 160 countries, we understand your local context and are here to guide you so you can provide your learners with everything they need to prepare for Cambridge IGCSE.

Quality management

We are committed to providing exceptional quality. In line with this commitment, our quality management system for the provision of international education programmes and qualifications for students aged 3 to 19 is independently certified as meeting the internationally recognised standard, ISO 9001:2015.
Learn more at **www.cambridgeinternational.org/about-us/our-standards/**

School feedback: ‘While studying Cambridge IGCSE and Cambridge International A Levels, students broaden their horizons through a global perspective and develop a lasting passion for learning.’

Feedback from: Zhai Xiaoning, Deputy Principal, The High School Affiliated to Renmin University of China

Cambridge International Education, The Triangle Building, Shaftesbury Road, Cambridge, CB2 8EA, United Kingdom
t: +44 (0)1223 553554 email: info@cambridgeinternational.org www.cambridgeinternational.org