Diploma Programme

# Computer science guide

First assessment 2027

International Baccalaureate®
Baccalauréat International
Bachillerato Internacional

Diploma Programme

# Computer science guide

First assessment 2027

International Baccalaureate®
Baccalauréat International
Bachillerato Internacional

# Diploma Programme
# Computer science guide

Published February 2025

Published by the International Baccalaureate Organization, a not-for-profit educational foundation of Rue du Pré-de-la-Bichette 1, 1202 Genève, Switzerland.
Website: ibo.org

## IB mission statement

The International Baccalaureate aims to develop inquiring, knowledgeable and caring young people who help to create a better and more peaceful world through intercultural understanding and respect.

To this end the organization works with schools, governments and international organizations to develop challenging programmes of international education and rigorous assessment.

These programmes encourage students across the world to become active, compassionate and lifelong learners who understand that other people, with their differences, can also be right.

# IB learner profile

**The aim of all IB programmes is to develop internationally minded people who, recognizing their common humanity and shared guardianship of the planet, help to create a better and more peaceful world.**

## As IB learners we strive to be:

### INQUIRERS
We nurture our curiosity, developing skills for inquiry and research. We know how to learn independently and with others. We learn with enthusiasm and sustain our love of learning throughout life.

### KNOWLEDGEABLE
We develop and use conceptual understanding, exploring knowledge across a range of disciplines. We engage with issues and ideas that have local and global significance.

### THINKERS
We use critical and creative thinking skills to analyse and take responsible action on complex problems. We exercise initiative in making reasoned, ethical decisions.

### COMMUNICATORS
We express ourselves confidently and creatively in more than one language and in many ways. We collaborate effectively, listening carefully to the perspectives of other individuals and groups.

### PRINCIPLED
We act with integrity and honesty, with a strong sense of fairness and justice, and with respect for the dignity and rights of people everywhere. We take responsibility for our actions and their consequences.

### OPEN-MINDED
We critically appreciate our own cultures and personal histories, as well as the values and traditions of others. We seek and evaluate a range of points of view, and we are willing to grow from the experience.

### CARING
We show empathy, compassion and respect. We have a commitment to service, and we act to make a positive difference in the lives of others and in the world around us.

### RISK-TAKERS
We approach uncertainty with forethought and determination; we work independently and cooperatively to explore new ideas and innovative strategies. We are resourceful and resilient in the face of challenges and change.

### BALANCED
We understand the importance of balancing different aspects of our lives—intellectual, physical, and emotional—to achieve well-being for ourselves and others. We recognize our interdependence with other people and with the world in which we live.

### REFLECTIVE
We thoughtfully consider the world and our own ideas and experience. We work to understand our strengths and weaknesses in order to support our learning and personal development.

**The IB learner profile represents 10 attributes valued by IB World Schools. We believe these attributes, and others like them, can help individuals and groups become responsible members of local, national and global communities.**

# Contents

# Purpose of this publication

This publication is intended to guide the planning, teaching and assessment of the subject in schools. Subject teachers are the primary audience, although it is expected that teachers will use the guide to inform students and parents about the subject.

This guide can be found on the subject page of the Programme Resource Centre at resources.ibo.org, a password-protected IB website designed to support IB teachers. It can also be purchased from the IB store at store.ibo.org.

## Additional resources

Additional publications such as specimen papers and markschemes, teacher support materials, subject reports and grade descriptors can also be found on the Programme Resource Centre. Past examination papers as well as markschemes can be purchased from the IB store.

Teachers are encouraged to check the Programme Resource Centre for additional resources created or used by other teachers. Teachers can provide details of useful resources, for example: websites, books, videos, journals or teaching ideas.

## Acknowledgement

The IB wishes to thank the educators and associated schools for generously contributing time and resources to the production of this guide.

| First assessment 2027 |
|---|

# The Diploma Programme

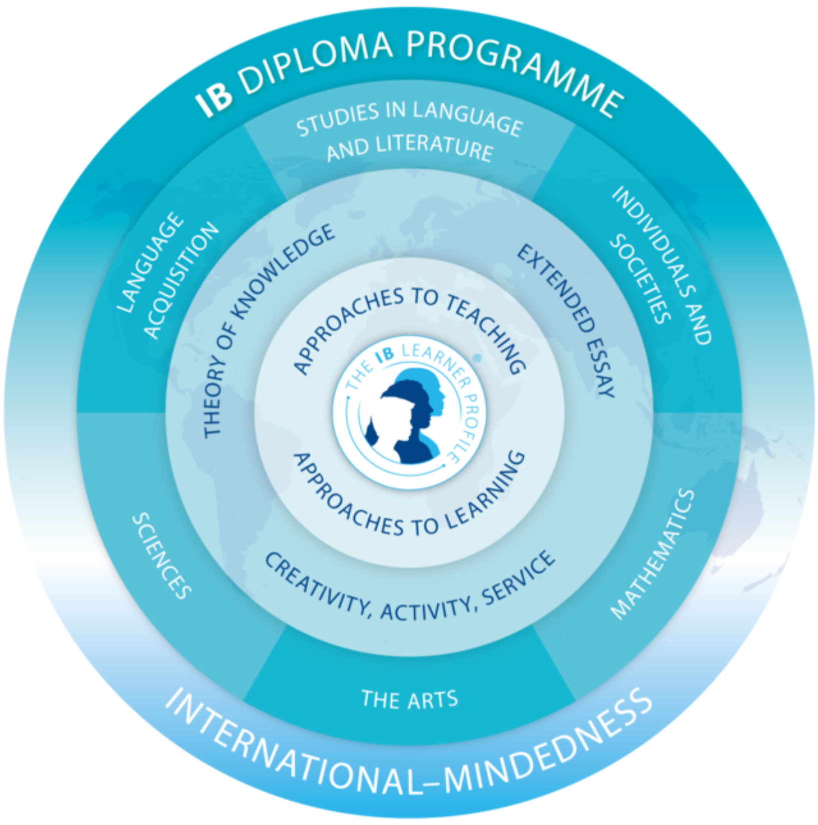
The Diploma Programme (DP) is a rigorous pre-university course of study designed for students in the 16 to 19 age range. It is a broad-based two-year course that aims to encourage students to be knowledgeable and inquiring, but also caring and compassionate. There is a strong emphasis on encouraging students to develop intercultural understanding, open-mindedness, and the attitudes necessary for them to respect and evaluate a range of points of view.

## The Diploma Programme model

The course is presented as six academic areas enclosing a central core (see figure 1). It encourages the concurrent study of a broad range of academic areas. Students study two modern languages (or a modern language and a classical language), a humanities or social science subject, an experimental science, mathematics and one of the creative arts. It is this comprehensive range of subjects that makes the DP a demanding course of study designed to prepare students effectively for university entrance. In each of the academic areas, students have flexibility in making their choices, which means they can choose subjects that particularly interest them and that they may wish to study further at university.

*Figure 1*

*Diploma Programme model*



## Choosing the right combination

**Students are required to choose one subject from each of the six academic areas, although they can, instead of an arts subject, choose two subjects from another area.** Normally, three subjects (and not more than four) are taken at higher level (HL), and the others are taken at standard level (SL). The IB recommends 240 teaching hours for HL subjects and 150 hours for SL. Subjects at HL are studied in greater depth and breadth than at SL.

At both levels, many skills are developed, especially those of critical thinking and analysis. At the end of the course, students' abilities are measured by means of external assessment. Many subjects contain some element of coursework assessed by teachers.

## The core of the Diploma Programme model

All DP students participate in the three course elements that make up the core of the model.

**Theory of knowledge** (TOK) is a course that is fundamentally about critical thinking and inquiry into the process of knowing rather than about learning a specific body of knowledge. The TOK course examines the nature of knowledge and how we know what we claim to know. It does this by encouraging students to analyse knowledge claims and explore questions about the construction of knowledge. The task of TOK is to emphasize connections between areas of shared knowledge and link them to personal knowledge in such a way that an individual becomes more aware of their own perspectives and how they might differ from others.

In TOK, students explore the means of producing knowledge within the core theme of "knowledge and the knower" as well as within various optional themes (knowledge and technology, knowledge and politics, knowledge and language, knowledge and religion, and knowledge and indigenous societies) and areas of knowledge (the arts, natural sciences, social sciences, history and mathematics). The course also encourages students to make comparisons between different areas of knowledge and reflect on how knowledge is arrived at in the various disciplines, what the disciplines have in common and the differences between them.

**Creativity, activity, service** (CAS) is at the heart of the DP. The emphasis in CAS is on helping students to develop their own identities, in accordance with the ethical principles embodied in the IB mission statement and the IB learner profile. It involves students in a range of activities alongside their academic studies throughout the DP. The three strands of CAS are creativity (arts, and other experiences that involve creative thinking), activity (physical exertion contributing to a healthy lifestyle) and service (an unpaid and voluntary exchange that has a learning benefit for the student). Possibly, more than any other component in the DP, CAS contributes to the IB's mission to create a better and more peaceful world through intercultural understanding and respect.

The **extended essay** (EE) offers the opportunity for IB students to investigate a topic of special interest, in the form of a 4,000-word piece of independent research. The area of research undertaken is chosen from one of the students' six DP subjects, or in the case of the interdisciplinary pathway, two subjects, and acquaints them with the independent research and writing skills expected at university. This leads to a major piece of formally presented, structured writing, in which ideas and findings are communicated in a reasoned and coherent manner, appropriate to the subject or subjects chosen. It is intended to promote high-level research and writing skills, intellectual discovery and creativity. An authentic learning experience, it provides students with an opportunity to engage in personal research on a topic of choice, under the guidance of a supervisor.

## Approaches to learning and approaches to teaching

Approaches to learning and approaches to teaching across the DP refers to deliberate strategies, skills and attitudes that permeate the learning and teaching environment. These approaches and tools, intrinsically linked with the learner profile attributes, enhance student learning and assist student preparation for DP assessment and beyond. The aims of approaches to learning and approaches to teaching in the DP are to:

- empower teachers as teachers of learners as well as teachers of content
- empower teachers to create clearer strategies for facilitating learning experiences in which students are more meaningfully engaged in structured inquiry and greater critical and creative thinking
- promote both the aims of individual subjects (making them more than course aspirations) and linking previously isolated knowledge (concurrency of learning)
- encourage students to develop an explicit variety of skills that will equip them to continue to be actively engaged in learning after they leave school, and to help them not only obtain university admission through better grades but also prepare for success during tertiary education and beyond
- enhance further the coherence and relevance of the students' DP experience
- allow schools to identify the distinctive nature of a DP education, with its blend of idealism and practicality.

The five approaches to learning (developing thinking skills, social skills, communication skills, self-management skills and research skills) along with the six approaches to teaching (teaching that is inquiry-based, conceptually focused, contextualized, collaborative, differentiated and informed by assessment) encompass the key values and principles that underpin IB pedagogy.

## The IB mission statement and the IB learner profile

The DP aims to develop in students the knowledge, skills and attitudes they will need to fulfil the aims of the IB, as expressed in the organization's mission statement and the learner profile. Learning and teaching in the DP represent the reality in daily practice of the organization's educational philosophy.

## Academic integrity

Academic integrity in the DP is a set of values and behaviours informed by the attributes of the learner profile. In teaching, learning and assessment, academic integrity serves to promote personal integrity, engender respect for the integrity of others and their work, and ensure that all students have an equal opportunity to demonstrate the knowledge and skills they acquire during their studies.

All coursework—including work submitted for assessment—is to be authentic, based on the student's individual and original ideas with the ideas and work of others fully acknowledged. Assessment tasks that require teachers to provide guidance to students or that require students to work collaboratively must be completed in full compliance with the detailed guidelines provided by the IB for the relevant subjects.

For further information on academic integrity in the IB and the DP, please consult the IB publications *Academic integrity policy, Effective citing and referencing, Diploma Programme: From principles into practice* and section "B1 General regulations: Diploma Programme" in the *Diploma Programme Assessment procedures* (updated annually). Specific information regarding academic integrity as it pertains to external assessment and internal assessment (IA) components of this subject can be found in this guide.

## Acknowledging the ideas or work of another person

Coordinators and teachers are reminded that students must acknowledge all sources used in work submitted for assessment. The following is intended as a clarification of this requirement.

DP students submit work for assessment in a variety of media that may include audiovisual material, text, graphs, images and/or data published in print or electronic sources. If a student uses the work or ideas of another person, the student must acknowledge the source using a standard style of referencing in a consistent manner. A student's failure to acknowledge a source will be investigated by the IB as a potential breach of regulations that may result in a penalty imposed by the IB Final Award Committee.

The IB does not prescribe which style(s) of referencing or in-text citation should be used by students; this is left to the discretion of appropriate faculty/staff in the student’s school. The wide range of subjects, three response languages and the diversity of referencing styles make it impractical and restrictive to insist on particular styles. In practice, certain styles may prove most commonly used, but schools are free to choose a style that is appropriate for the subject concerned and the language in which students’ work is written. Regardless of the reference style adopted by the school for a given subject, it is expected that the minimum information given includes: name of author, date of publication, title of source, and page numbers as applicable.

Students are expected to use a standard style and use it consistently so that credit is given to all sources used, including sources that have been paraphrased or summarized. When writing text students must clearly distinguish between their words and those of others by the use of quotation marks (or other method, such as indentation) followed by an appropriate citation that denotes an entry in the bibliography. If an electronic source is cited, the date of access must be indicated. Students are not expected to show faultless expertise in referencing, but are expected to demonstrate that all sources have been acknowledged. Students must be advised that audiovisual material, text, graphs, images and/or data published in print or in electronic sources that is not their own must also attribute the source. Again, an appropriate style of referencing/citation must be used.

## Learning diversity and learning support requirements

Schools must ensure that equal access arrangements and reasonable adjustments are provided to students with learning support requirements that are in line with the IB publications *Access and inclusion policy* and *Learning diversity and inclusion in IB programmes: Removing barriers to learning*.

The publications *Meeting student learning diversity in the classroom* and *The IB guide to inclusive education: a resource for whole school development* are available to support schools in the ongoing process of increasing access and engagement by removing barriers to learning.

# Nature of the subject

## What is computer science?

Computer science is no more about computers than astronomy is about telescopes.

(Dijkstra, 1967)

Computer science is the study of computers and computational systems. It covers a range of topics related to the theoretical aspects of computing, including algorithms and software design, and the application of computer science to solve practical problems. Computer science is distinct from the natural sciences in that it does not rely on hypothesis and experimentation. Computer science can be considered to be cross-disciplinary, as it draws from a range of disciplines, especially mathematics.

## How did computer science develop?

The foundational concepts of computer science have evolved over centuries, shaped by the ground-breaking contributions of luminaries such as Al-Khwarizmi, Ada Lovelace, Alan Turing, John von Neumann, Grace Hopper and Tim Berners-Lee. The progress of computer science has been a journey from rudimentary mechanical devices to today's advanced electronic machines and sophisticated software systems. Beginning with ancient tools like the abacus for basic arithmetic operations, computing transitioned through ground-breaking inventions such as Charles Babbage's steam-powered Analytical Engine in the 19th century to Alan Turing's theoretical universal computing machine. This in turn inspired John von Neumann and others on the Scientific Advisory Board of the United States government to produce the first working computer, ENIAC. These beginnings have led to a world where computing and computer science applications are essential elements of day-to-day life.

## What is computational thinking?

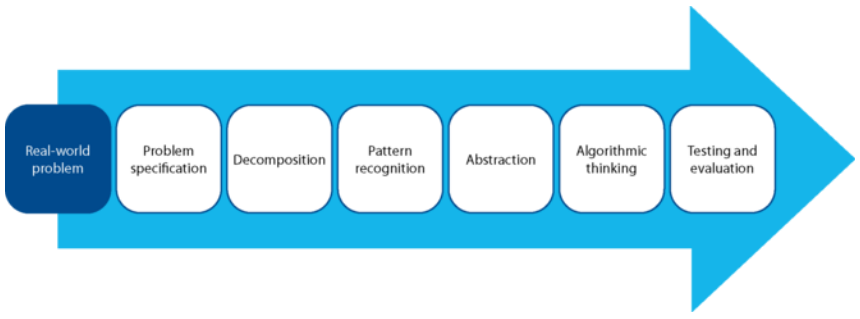
Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out.

(Wing, 2014)

In the DP computer science course, students develop computational thinking, a problem-solving technique that can be applied to everyday challenges. Computational thinking is a crucial skill set in the modern digital age, allowing us to tackle problems by leveraging the power of computer processes. As highlighted in the quote by Jeannette Wing, computational thinking does not necessarily involve programming. Rather, it focuses on understanding and solving problems in a manner that a computer could execute.

*Figure 2*

*Computational thinking process*



The key components of computational thinking begin with a problem and then comprise the following.

- **Problem specification:** Clearly defining and understanding the nature of a problem, the limitations and its scope for establishing solution goals.
- **Decomposition:** Breaking down a complex problem or system into smaller, more manageable parts.
- **Pattern recognition:** Identifying similarities to other problems, or identifying recurring elements in the system, to make predictions and develop algorithms.
- **Abstraction:** Focusing on essential features and high-level ideas, whilst removing unnecessary detail, to develop a system overview.
- **Algorithmic thinking:** Developing a step-by-step series of instructions for solving a particular problem.
- **Testing and evaluation:** Assessing a potential solution against the initial expected goals to determine the effectiveness of the solution, or the adjustments needed to solve the problem completely.

## Skills in the study of computer science

The goal of computer science is to solve meaningful problems—problems for which an obvious algorithm is not available. To solve problems, the practical skills of computer science need to be developed as part of the computational thinking process.

The skills of computer science can be classified as both technical skills and personal skills. Technical skills are centred on algorithmic thinking and computer programming. Personal skills are focused on the ability to collaborate, think critically, think imaginatively and engage in the process of computational thinking. These personal skills are covered in more detail in the sections “Computer science and the IB learner profile” and “Approaches to learning and approaches to teaching in computer science”.

### Computational thinking and course content

The focus of the DP computer science course is computational thinking. Wherever possible, learning and teaching should take place with computational thinking in mind. This will develop students’ understanding of this concept and develop their ability to apply it. The focus on computational thinking also allows students to develop the skills needed for the IA task—the computational solution. Examples of units of work and individual lesson plans are given in the *Computer science teacher support material*.

### Practical skills in computer science

Integral to the student experience of computer science is the learning that takes place through the practical application of computational thinking within the classroom or in an external context. Introducing a topic by raising problem scenarios that can be addressed using computational thinking develops in students a practical methodology with a focus on problem-solving. Progressive increasing complexity in these scenarios then promotes the development, improvement and consolidation of the practical skills of algorithmic thinking and computer programming.

Students are encouraged to improve their practical skills of algorithmic thinking and programming through the conceptual and methodological perspectives of computational thinking. Students should also be encouraged to apply these skills where possible to problems in their personal, academic or social environment that they consider significant and challenging. Examples are “to do” lists, study calendars, personal budget tools, robotics projects, puzzles, computer games and mobile applications. This process of inquiry is central to the computer science course.

## Algorithmic thinking

An algorithm is a precise sequence of well-defined instructions designed to solve specific problems or perform tasks. Algorithms are vital in computer science and are a cornerstone of software development, underpinning data manipulation, search, sorting and optimizing. They allow computers to execute tasks accurately and efficiently.

Algorithmic thinking—the process of developing algorithms—is a critical component of computational thinking. The algorithmic thinking process fosters the capacity to abstract problems and formulate algorithms, whether as code or in other forms of representation.

The process of algorithmic thinking has four key steps.

1. **Problem exploration and parameter variation:** Analyse the problem by working through examples with different parameters to uncover patterns and inform algorithm design.
2. **Detailed step-by-step documentation:** Record the steps taken for the different parameters to create a first draft of the algorithm.
3. **Pattern recognition and generalization:** Identify recurring patterns across different parameters to develop a set of steps applicable to a general parameter, converting the specific process in the first draft into a general algorithm.
4. **Testing and validation:** Finalize the algorithm by subjecting it to rigorous testing with a range of input parameters to ensure consistent and correct results. This is an iterative process, much like the scientific method, that never guarantees universal applicability. However, a good testing strategy will allow the algorithm to be refined and validated. This refinement will take into account the algorithm’s performance in terms of time (run time) and space (memory requirements). Different algorithms can solve the same problem, but there may be differences in terms of efficiency.

The set of steps described above assumes that algorithmic thinking always involves identifying a pattern, generalizing it to form an algorithm, and then testing the algorithm. This is quite similar to the scientific method of observation, hypothesis and experimentation. However, it is important to note that the validity of algorithms can also be judged by the logical sequence of their instructions and the validity of their mathematical calculations.

In the process of developing these skills, students should be able to write technical documentation, including constructing different types of diagrams that illustrate the functional requirements and internal structures of the software solution.

## Computer programming languages

The key application of algorithmic thinking to solve problems in computer science is computer programming. Students should be able to apply their skills of algorithmic thinking by developing a relevant computer program or software application. The skills of computer programming are a fundamental part of the computer science course and students will need to develop these skills in the relevant programming language: **Java** or **Python**. Course material is not specific to either programming language.

Additionally, students will evaluate and design the data structures needed for a solution, such as variables with correct data types, static/dynamic data structures, classes, databases and I/O files. Students should be able to design efficient methods that will be used for manipulating the data within a software solution. To evaluate the effectiveness of a computer program, exception handling, debugging and testing techniques will also be an essential part of the computer programming skill set.

Examination questions that include code will have two versions of the question, one in Java and one in Python. These questions will be comparable, assessing the same topics from the syllabus with the same contexts, and assessing the same assessment objectives (AO).

The Python programming language has a number of built-in functions that can help with finding solutions to common computational challenges. The use of functions such as sort, pop, len, max and min can be very useful when programming algorithms and their use is encouraged in tasks such as the IA computational solution. However, in order to assess the full range of students' abilities, certain questions in the examination will prohibit specific built-in functions. These instances will be clearly indicated in the examination papers.

## What is the impact of computer science?

Computational thinking is a process that can be transferred to, and applied in, many different disciplines. These include mathematics and the natural sciences, but also disciplines outside the sciences where computation plays a role in deriving results. In computer science, the computational thinking process is used to solve problems by applying the skills of algorithmic thinking and computer programming.

The problems that computer scientists solve often respond to challenges in the real world, so their solutions can have a profound impact on society, transforming many aspects of our daily lives, businesses, industries and government. These societal changes come with challenges, such as issues of privacy, digital addiction and the digital divide.

As society continues to evolve with technology, it is crucial to address these concerns to ensure that the benefits of computer science are shared equitably and its potential pitfalls are mitigated. Computer scientists should be aware that when they abstract a problem to find a computational solution, this solution may well have far-reaching applications and consequences in the real world, significantly affecting the everyday lives of millions.

As with all disciplines, there are ethical issues associated with computer science, particularly when it is applied to practical uses. Computer scientists need to consider that solutions and developments that may not initially seem to raise ethical questions could yet be applied in contexts that are ethically problematic.

Ethics is addressed directly in the topic on machine learning (A4.4.1); meanwhile other topics address ethical issues through linking questions and TOK questions.

## Distinction between SL and HL

Students at both SL and HL study the following parts of the course.

- The nature of computer science, with computational thinking as an organizing concept
- An understanding of computer science through the practical use of a programming language (Java or Python), and the object-oriented programming (OOP) paradigm
- The application of Structured Query Language (SQL) to extract and manipulate data
- One IA task: The computational solution
- A case study focusing on emerging technologies
- The collaborative sciences project

The SL and HL courses both provide students with an understanding of computer science through the organizing concept of computational thinking and the practical use of skills in computer science to solve problems.

The distinction between SL and HL is in both breadth and depth. The SL course has a recommended 150 teaching hours and the HL course has a recommended 240 hours.

- SL students study seven topics on concepts and problem-solving in computer science, whereas HL students study an extra topic (abstract data types) with additional HL-only material within the seven SL topics.

- HL students study the topics to increased breadth, resulting in increased networked knowledge. HL requires the student to make more connections between diverse areas of the syllabus, thereby also increasing the depth of coverage.
- HL students also conduct deeper research in the case study, reflected in the extra two challenge questions in the case study for HL, additional recommended teaching hours, and additional time for paper 1.

The HL course provides a solid foundation for the study of computer science or related subjects at university level.

# Computer science and the DP core

## Computer science and theory of knowledge

The TOK course plays a special role in the DP by providing opportunities for students to reflect on the TOK framework of scope, perspectives, methodologies and ethical implications of knowledge. These reflections should also take place in computer science by exploring knowledge questions relevant to the subject. This is valuable for students to deepen their understanding and make connections between disciplines.

The fact that computer science includes the word "science" does not mean that it can be easily compared to the natural sciences. Knowledge of computer science is not the same as knowledge of computers. As an abstract form of knowledge, computer science is similar to mathematics. Like mathematics, the applications of computer science are far-reaching and significantly affect the world we live in.

The following TOK questions focus on the **scope** of knowledge in computer science.

- How does knowledge in computer science compare to knowledge in the five different "areas of knowledge" in TOK?
- How does our understanding of algorithms influence our perception of decision-making processes in both machines and humans?

There are many perspectives on knowledge within computer science. Some view it as an extension of mathematics, some see it as an empirical science, while others consider it a branch of engineering. Indeed, computer science is a multifaceted discipline and its practitioners often bring their unique viewpoints to the table. Many people interact with computers daily, but only a fraction truly understand the underlying principles of computer science. As a result, perspectives on what constitutes knowledge in this field can vary greatly.

The following TOK questions focus on different **perspectives** of knowledge in computer science.

- How do virtual reality and augmented reality challenge or extend our sense perception?
- How does the dominance of English in programming languages impact learning and accessibility for people whose first or best language is not English?

Computer science does indeed have "science" in its name, but its methodology often diverges from the traditional scientific method found in fields like biology, physics or chemistry. For example, computer scientists might design algorithms and then prove their correctness using mathematical methods. However, in machine learning, the evaluation of algorithmic performances can be seen as an experimental process.

The nature of computer science requires the use of deductive reasoning to find abstract programming solutions to real-world problems, and inductive reasoning to test for the best solution. Knowledge in computer science is often manifested through its application in technology and can be seen as an engineering solution as well as a pure form of knowledge. This is similar to physics, in that there is both a theoretical and experimental element to the knowledge.

The following TOK knowledge questions focus on **methodology** in computer science.

- How does knowledge in computer science develop?
- How do the designs of social media algorithms manipulate human emotions?

The tools and skills of computer science raise a number of ethical issues. Examples of these are computer hacking, intellectual property theft, privacy, data collection and algorithmic bias. The acceptance that there are ethical issues specific to computer science, not just its applications, is an important reflection.

The following TOK knowledge questions focus on the **ethics** of knowledge in computer science.

- To what extent do computer programs include the bias(es) of the computer programmer?

- How can we ensure that computer programs are limited to ethically acceptable applications?

More information on TOK can be found in the *Theory of knowledge guide*.

## Computer science and creativity, activity, service

The CAS component of the DP core provides many opportunities for students to link concepts in computer science to practical experiences. Teachers can highlight how knowledge and understanding developed through the course might inform meaningful experiences.

Examples of how CAS experiences can be linked to computer science include:

- teaching—offering introductory programming lessons to students in the school
- collaboration—organizing, hosting or participating in a school hackathon
- advocacy and solution development—creating a website or software solution to address a local concern as a force for positive change.

More information on CAS can be found in the *Creativity, activity, service guide*.

## Computer science and the extended essay

Computer science is a popular choice for the EE. The research skills developed by students undertaking an EE in computer science not only benefit them in their study of DP computer science, but also prepare them for study in computer science and other subjects beyond the DP.

Examples of research questions for EEs in computer science include the following.

- To what extent does the application of the Coppersmith–Winograd matrix multiplication algorithm improve convolution neural network (CNN) training times?
- To what extent do the differences between multithreading and asynchronous programming in Java affect the performance of computationally intensive tasks versus input–output intensive tasks?
- Does additional training of the discriminator in a Generative Adversarial Network (GAN) lead to a strong correlation between the discriminator's accuracy and the inception score of the generator's outputs?

It is important to recognize the differences between an EE in computer science and the IA component of the course. Whereas the IA is a structured assessment designed to evaluate a student's understanding and application of computational thinking, course material and programming skills, an EE in computer science provides students with an opportunity to undertake deeper research into an area of interest to them. A key objective of the EE is to engage students in independent research through an in-depth study of a question relating to one aspect of computer science. The emphasis is on research, analysis and argumentation. The EE is assessed externally, while the IA is internally assessed and externally moderated.

Teachers who are acting as supervisors for students completing EEs in computer science should ensure they refer to the *Extended essay guide* for both generic and subject-specific guidance.

# Computer science and international-mindedness

International-mindedness lies at the core of the IB's philosophy, representing a commitment to fostering global engagement, multilingualism and intercultural understanding. Computer science is, by its very nature, internationally minded. It transcends national borders through the exchange of ideas and methodologies, and hence has the potential to foster intercultural understanding. The rapid advances in information and communication technologies in recent decades have further accelerated this global interchange.

Several facets of modern computer science are examples of its international dimension.

- **Open-source movements**—open-source platforms invite contributors throughout the world to enhance software through shared knowledge and collective effort.
- **Online collaboration**—development platforms and knowledge repositories provide arenas for computer scientists worldwide to share, discuss and iterate on ideas through global teamwork.
- **Social networking**—platforms designed by computer scientists have redefined global communication, turning a wide of range interactions between people across the world into an everyday occurrence.

Computer scientists often grapple with international challenges, crafting computational solutions aimed at creating a safer, more sustainable world. This proactive approach to solving real-world problems, whether local or global, is integral to the ethos of the IB computer science curriculum. Through the DP computer science course, students learn to become both proficient and globally conscious citizens, ready to make meaningful contributions in an interconnected world.

# Computer science and the IB learner profile

The learner profile is a set of attributes that the IB promotes as an ideal for its students. Each box under the section “Attributes of the IB learner profile” provides an example of how the attribute could be modelled by learners and teachers during the DP computer science course, based on the following approach.

| Example attribute |
|---|
| • How students can best embody the attribute with reference to computer science.<br>• Advising teachers on possible routes to develop the attribute in the classroom.<br>• Practical ways in which students demonstrate the attribute in the process of “doing” computer science. |

## Attributes of the IB learner profile

| Inquirer |
|---|
| • Inquirers are curious, actively use research skills and ponder real-world problems; the potential solutions they arrive at use the knowledge and skills they have learned in computer science.<br>• Teachers facilitate skills development and promote inquiry; they provide students with opportunities to seek out problems and search for solutions.<br>• Students use their inquiry skills to extend their knowledge and skills in computer science through inquiry and research. |

| Knowledgeable |
|---|
| • Students explore concepts, ideas and issues to broaden and deepen their understanding of computational thinking and to develop skills in programming.<br>• Teachers access a variety of resources to develop and improve their students’ understanding and skills.<br>• Students apply their knowledge to unfamiliar contexts, and make connections between concepts and solutions to problems, to illustrate their understanding of computer science. |

| Thinker |
|---|
| • Students solve complex problems and reflect on the effectiveness of their computational thinking.<br>• Teachers provide opportunities for students to analyse their approaches and methods critically and to deepen their understanding of computer science, allowing them to be creative in finding solutions to problems.<br>• Students practise reasoning and computational thinking by iterating through a cycle of formulating, testing, debugging and improving solutions, through consideration of outcomes. |

**Communicator**

- Students collaborate effectively with others and use a variety of modes of communication to express their ideas and solution strategies.
- Teachers facilitate group work to promote collaborative computational thinking and programming, and to encourage successful communication.
- Students demonstrate effective communication in collaborative activities by actively listening and sharing ideas.

**Principled**

- Students take responsibility for their work, promote shared values and act in an ethical manner, including respecting intellectual property constraints and licensing terms.
- Teachers provide opportunities to model principled behaviour, including acknowledging the work of others and citing sources.
- Students appreciate the importance of integrity in data collection, the appropriate use of personal data, and recognize the possibility of algorithm bias in their programs.

**Open-minded**

- Students accept that there are different perspectives on a problem and that different solutions can exist.
- Teachers provide examples of problems that have a variety of possible solutions that can meet the underlying needs of the end user in different ways.
- Students remain open to alternative perspectives on how a problem can be framed or solved.

**Caring**

- Students act to ensure that solutions to problems—and even identifying a problem in the first place—are formulated to improve the lives of others.
- Teachers draw attention to how daily choices have consequences by challenging students to adopt good practice and providing support to help fellow students.
- Students connect solutions to real-world problems to global challenges. The collaborative sciences project allows students to support one another, fostering group success.

**Risk-taker**

- Risk-takers seek new opportunities to develop their learning, explore new problem-solving approaches, and thrive when looking for new ways to approach a challenge.
- Teachers provide support and guidance for students, encouraging them to explore new techniques or methods of learning.
- Students are prepared to experiment with new approaches, information or data that may affect their solution to a problem. They understand that this is a step forward in their understanding.

**Balanced**

- Balanced learners look holistically at all aspects of their development; they ensure that all their tasks are given appropriate attention without focusing on one to the detriment of others.
- Teachers encourage students to approach solutions with a balanced, unbiased perspective, using computational thinking.

| Balanced |
|---|
| • Students organize their own time effectively, managing their tasks and projects successfully. They give themselves time to complete all parts of their learning without negatively impacting the emotional and social aspects of their lives. |

| Reflective |
|---|
| • Reflective learners consider why and how they achieve success, and how they could change their approach when learning is difficult.<br>• Teachers provide opportunities for students to continually review strategies and approaches to problem-solving, in order to improve their algorithmic thinking and programming skills.<br>• Students continually refine and connect their knowledge by reflecting on their evolving understanding. |

# Approaches to learning and approaches to teaching in computer science

## The approaches to learning framework

### What are the approaches to learning skills and why do we teach them?

The approaches to learning framework seeks to develop in students affective, cognitive and metacognitive skills that will support their learning processes during and beyond their IB experience. The development of approaches to learning skills is closely connected with the IB learner profile attributes and therefore helps to advance the IB mission. The skills are an integral part of IB learning and teaching that should be developed across the whole programme—it is not expected that a single course should ever address all of them.

### How are they organized?

The approaches to learning framework for IB programmes consists of five general skill categories: thinking skills, communication skills, social skills, research skills and self-management skills. Each of these categories covers a broad range, as shown by the examples presented in the following table. The skill categories are closely linked and interrelated and therefore individual skills may be relevant to more than one category.

### How do we teach them?

Approaches to learning skills can be learned and taught, improved with practice and developed incrementally. The table illustrates, through a number of examples, how the computer science course can support approaches to learning skills development. The examples shown are not exhaustive. Teachers are encouraged to adapt them for use in their school context and collaboratively identify further examples of approaches to learning skills development. Further information on the framework and strategies for the development of the approaches to learning skills can be found in the *Computer Science teacher support material* and the DP *Approaches to teaching and learning website*.

### Skills and development

| Skill category | Examples of skills development in the classroom |
|---|---|
| **Thinking skills** | • Be curious about the type of problems that can be solved using computing.<br>• Analyse scenarios that can be framed as problems in computing.<br>• Decompose problems into sub-problems.<br>• Recognize data patterns.<br>• Use abstraction to explain the structure of a problem by ignoring non-essential features.<br>• Design algorithms to solve problems.<br>• Test solutions and look for improvements.<br>• Evaluate the success of computational problem-solving.<br>• Analyse data gathered from research.<br>• Combine ideas to create new understandings. |

| Skill category | Examples of skills development in the classroom |
|---|---|
| **Communication skills** | • Practise active listening skills.<br>• Reflect on the needs of the audience when translating technical concepts into general understandings.<br>• Develop skills in written and diagrammatic communication relevant to computer science (e.g. system overview diagrams, flowcharts, UML diagrams).<br>• Communicate solutions with other computer scientists.<br>• Use terminology, symbols and programming styles consistently and correctly.<br>• Present data appropriately to recognize data patterns.<br>• Communicate constructive criticism appropriately when evaluating the solutions of other computer scientists.<br>• Use effective comments and correct syntax.<br>• Use efficient computational instructions and conventions to increase readability and aid debugging. |
| **Social skills** | • Recognize the importance of collaboration in computer science (e.g. open-source movements).<br>• Work in teams to achieve common goals in different types of collaborative environments.<br>• Assign and accept specific roles during collaborative activities, recognizing their significance in the process of computational thinking.<br>• Appreciate the diverse talents and needs of others.<br>• Resolve conflicts during collaborative work.<br>• Actively seek and consider the perspective of others.<br>• Constructively assess the contribution of peers. |
| **Research skills** | • Use human, printed and digital resources effectively.<br>• Evaluate information sources for accuracy, bias, credibility and relevance.<br>• Discuss the importance of academic integrity and acknowledge the ideas of others.<br>• Use a consistent, standard method of referencing and citation.<br>• Compare, contrast and validate information. |
| **Self-management skills** | • Develop organizational skills through goal setting and planning.<br>• Manage tasks with effective time management.<br>• Break down major tasks into a sequence of stages.<br>• Be punctual and meet deadlines.<br>• Regard setbacks as part of the process towards solutions.<br>• Set learning goals and adjust them in response to experience.<br>• Seek and act on feedback. |

## Approaches to teaching

The teaching of IB programmes, based on a constructivist and student-centred approach, models its essential pedagogical structure on six principles.

- **Inquiry** that strengthens creativity, autonomy, communication skills and organizational skills

- **Conceptual understanding** that helps students address complex problems and transition to higher-order thinking
- Teaching developed in **local and global contexts** that contributes to contextualized and meaningful learning, and the development of international-mindedness
- **Teamwork and collaboration** that supports the sense of collaboration among learners and fosters the principle of collective responsibility
- **Teaching** that affirms a pedagogical balance, asserts identity and creates **differentiated learning** opportunities
- **Teaching informed by assessment** that validates curricular objectives and guarantees efficient learning

## Inquiry in computer science

Inquiry begins by identifying problems that can be solved using the conceptual framework of computational thinking (figure 2) and the skills of computer science. This process requires problem-solving skills and critical thinking skills. Inquiry supports students to become self-dependent and collaborative, thus fostering their natural curiosity.

Students should be encouraged to develop investigations to support their learning through open-ended inquiry, with a focus on problem specification, abstraction, algorithmic thinking and evaluation. Open-ended investigation of problems based on their own inquiry allows students to understand the challenges of decomposing problems, constructing system requirements and developing algorithms. Students who are engaged in this process are more likely to reflect critically and evaluate outcomes successfully.

## Conceptual learning

Concept-based learning and teaching is encouraged across the continuum of IB programmes. Concepts are mental representations of categories. They are constructed, modified and activated by the learner through learning experiences, and do not exist in isolation but are interrelated.

Conceptual understanding is the result of a non-linear and continuous process of evolving understanding by adapting prior knowledge, and identifying and dispelling misconceptions. It consists of establishing connections between prior and new knowledge in order to build understanding and become aware of this network of knowledge.

Concepts vary in their level of abstraction and scope. They can:

- organize ideas that are applicable in many contexts and have relevance both within and across subject areas
- provide a deep understanding of specific fields of knowledge and help to organize knowledge further, as well as reveal connections between different areas of the subject.

For example, consider the following sequence of three concepts.

**Data structures** > **Algorithms** > **Optimization**

Optimization is a component in the understanding of algorithms, which in turn helps to develop an understanding of the broader concept of data structures in computer science.

Another example is as follows.

**Programming syntax** > **Software logic** > **Application development**

Application development is a component in the understanding of software logic, which in turn helps to develop a more general understanding of the concept of programming syntax.

## Outcomes of a concept-based approach

By fostering conceptual understanding in computer science, students will be able to:

- identify problems that can be solved using computing

- decompose problems and use abstraction to make them accessible to an algorithmic solution
- apply concepts to existing and new contexts
- apply their conceptual understanding in computational thinking to test and evaluate solutions
- transfer the functional structure of a problem solved to other more complex problems.

## The syllabus and conceptual understanding

The structure of this computer science syllabus aims to promote concept-based learning and teaching through the organizing concept of computational thinking. The syllabus is intended to examine key concepts in computer science—computer fundamentals, networks, databases and machine learning—and then apply practical skills to support the computation thinking process to solve problems. The key practical skills are algorithmic thinking and programming.

There are two organizing structures for the syllabus reflected as two separate themes.

- Theme A: Concepts of computer science
- Theme B: Computational thinking and problem-solving

The themes of the course have been chosen to represent the connection between abstract ideas of how computing systems operate (theme A), and their application using the practical skills of computer science to solve problems through the process of computational thinking (theme B).

Each of these themes is subdivided into topics with guiding questions, recommended teaching hours for each level, and linking questions. The topics are divided into subtopics, each with learning statements that state the understandings that students should develop and further guidance.

The purpose of the guiding questions for each topic is to promote inquiry. They are therefore not straightforward and are best answered once the associated understandings have been acquired. Teachers and students are encouraged to create their own guiding questions based on the content of units of study.

The linking questions at the end of each topic are designed to facilitate connections. These links promote conceptual understanding of computer science as a network of complementary ideas that relate to each other and beyond—to understandings outside the field of computer science.

For example, when considering the processes and structures within a computer's central processing unit (CPU), students can start with the structure of the instruction set architecture, connect this to the execution of machine code instructions, and in turn connect this to the running of software applications. By looking at this process through the concept of computational efficiency, principles of algorithm optimization and resource management are also highlighted, which further connects to other areas of computer science.

Note that the linking questions found in the guide are not exhaustive. Students and teachers may well encounter other connections between understandings and concepts in the syllabus, leading to additional linking questions.

## Teaching computer science in context

Computer science has brought benefits to many fields of human experience and is an essential element of finding effective solutions to a range of global challenges. Teaching computer science in context therefore explores how computer science impacts our world in many different ways, promoting student interest, understanding and inquisitiveness.

Studying computer science enables constructive engagement with topical issues in computing. Studying its concepts in context will allow students to better understand issues such as privacy and how social networks can include algorithmic bias.

Teaching the course with reference to different scenarios and contexts supports the pedagogical principle of teaching in local and global contexts, as part of the approaches to teaching framework, and offers a number of advantages. First, it helps students relate their learning to genuine applications of computer science, highlighting its relevance to global issues as well as its significance in students' own contexts. Second, it develops an appreciation for the interaction between solutions in computer science and their implications, be they ethical, environmental or economic. Third, it helps to illustrate the computational thinking process underpinning the course.

The *Computer science teacher support material* highlights possible areas that could be visited throughout the course to stimulate problem-posing and solutions development for specific topics through particular contexts. Taking these and other related areas into consideration may help provide ideas for the computational solution, the collaborative sciences project, the TOK exhibition, CAS or an EE in computer science or world studies.

## Prior learning

Past experience demonstrates that students are able to study computer science at SL successfully with no background in the subject. Their approach to study, characterized by specific IB learner profile attributes, will be significant.

For students considering computer science at HL, some previous exposure to programming is recommended, although there is no intention to restrict access. HL demands a higher level of problem-solving skills and a greater ability to understand and manipulate abstract concepts. Students who enjoy mathematical problem-solving are well suited to studying HL computer science.

## Links to the Middle Years Programme

The Middle Years Programme (MYP) offers a coherent pedagogical framework for learning and teaching, alongside flexibility and choice regarding specific curriculum content.

MYP mathematics and design courses can expose students to the practical skills of computer programming and algorithmic thinking, and can therefore be a good basis for students to study DP computer science. It is also true that MYP mathematics and design courses can differ greatly between different MYP schools. Therefore, some students may not have been exposed to programming. But the principles underpinning the MYP curriculum—inquiry, problem-solving and critical thinking—will be of benefit to any student choosing to study DP computer science. The DP course requires students to become actively involved in and focused on problem-solving through the computational thinking process, building on the skills acquired during the MYP, especially in mathematics.

To successfully complete the DP computer science IA task (the computational solution), students are expected to create a solution to a specific problem of their own choosing using computational thinking. This extends the range of skills and attributes developed in MYP, including the ability to work independently and in collaboration with other students.

## Links to the Career-related Programme

In the Career-related Programme (CP), students study at least two DP subjects, a core consisting of four components and a career-related study, which is determined by the local context and aligned with student needs. The CP has been designed to add value to the student’s career-related studies. This provides the context for the choice of DP courses. Courses can be chosen from any group of the DP. It is also possible to study more than one course from the same group (for example, visual arts and film).

Computer science may be a beneficial choice for CP students considering careers in, for example, the technology, financial, manufacturing and telecommunications industries, and international business. The use of computers in work and everyday life is ubiquitous. This is commonly known as information technology. However, computer science is not the same as information technology, and is concerned with both how computers work and how computing can be used to solve real-world problems. In addition to skills in information technology, understanding the role of computer science in a career-related activity can be beneficial. The ability to use computational thinking and developing programming skills are important in many future-facing careers.

Computer science encourages the development of strong problem-solving skills, critical thinking and computational thinking, and a deep understanding of how digital systems work, all of which will help students prepare for the future global workplace. This in turn fosters the IB learner profile attributes that are transferable to the entire CP, providing relevance and support for the student’s learning.

For CP students, DP courses can be studied at SL or HL. Schools can explore opportunities to integrate CP students with DP students.

## Collaborative sciences project

The collaborative sciences project is an interdisciplinary sciences project, providing a worthwhile challenge to DP and CP students, addressing real-world problems that can be explored through the sciences. The nature of the challenge should allow students to integrate factual, procedural and conceptual knowledge developed through the study of their disciplines.

Through the identification and research of complex issues, students can develop an understanding of how interrelated systems, mechanisms and processes impact a problem. Students will then apply their collective understanding to develop solution-focused strategies that address the issue. With a critical lens, they will evaluate and reflect on the inherent complexity of solving real-world problems.

Students will develop an understanding of the extent of global interconnectedness between national, regional and local communities, which will empower them to become active and engaged citizens of the world. While addressing local and global issues, students will appreciate that the issues of today exist across national boundaries and can only be solved through collective action and international cooperation.

The collaborative sciences project supports the development of students' approaches to learning skills, including teambuilding, negotiation and leadership. It facilitates an appreciation of the environment, and the social and ethical implications of science and technology.

Full details of the requirements are in the *Collaborative sciences project guide*.

# Aims and objectives

## Computer science aims

The course enables students to:

1. develop conceptual understanding that allows connections to be made between different areas of the subject, and to other DP subjects
2. acquire and apply a body of knowledge, methods, tools and techniques that characterize computer science
3. analyse and evaluate solutions developed through computational thinking in a range of contexts
4. approach unfamiliar situations with creativity and resilience
5. use computational thinking to design and implement solutions to local and global problems
6. develop an appreciation of the possibilities and limitations of computer science
7. evaluate the impact of emerging technologies in computer science
8. communicate and collaborate effectively
9. develop awareness of the environmental, economic, cultural and social impact of computer science, its applications and ethical implications.

## Assessment objectives

The objectives for computer science are as follows.

**AO1** Demonstrate knowledge and understanding of:

- facts, concepts, principles and terminology in computer science
- appropriate methods, techniques and skills to solve problems using computational thinking.

**AO2** Apply and use:

- facts, concepts, principles and terminology in computer science
- appropriate methods, techniques and skills to solve problems using computational thinking
- appropriate methods to present information in computer science.

**AO3** Construct, synthesize, analyse and evaluate:

- problem specifications, system requirements, success criteria, testing strategies and programs
- appropriate techniques to solve a problem
- relevant data, information and technological explanations for solutions.

**AO4** Demonstrate the application of computational thinking skills to solve real-world problems using computer science solutions.

## Assessment objectives in practice

The assessment components align with the course aims, objectives and conceptual approach. This allows students to demonstrate their learning effectively, through varied tasks that are reliably and accurately marked or moderated by subject-specific educators and experts.

| Assessment objective | Which component addresses this assessment objective? | How is the assessment objective addressed? |
|---|---|---|
| **AO1** Demonstrate knowledge and understanding | Paper 1<br>Paper 2<br>IA: The computational solution | Students respond to a range of short- and extended-response questions.<br>Students develop a solution to an appropriate problem of their choice. |
| **AO2** Apply and use knowledge | Paper 1<br>Paper 2<br>IA: The computational solution | Students respond to a range of short- and extended-response questions.<br>Students develop a solution to an appropriate problem of their choice. |
| **AO3** Construct, synthesize, analyse and evaluate | Paper 1<br>Paper 2<br>IA: The computational solution | Students respond to a range of short- and extended-response questions.<br>Students develop a solution to an appropriate problem of their choice. |
| **AO4** Demonstrate the application of computational thinking skills to solve real-world problems using computer science solutions | IA: The computational solution | Students develop a solution to an appropriate problem of their choice. |

| Component | Approximate weighting of assessment objectives (%) | |
|---|---|---|
| | **AO1 + AO2** | **AO3** |
| Paper 1 | 50 | 50 |
| Paper 2 | 50 | 50 |
| IA | Covers AO1, AO2, AO3 and AO4 | |

# Syllabus outline

| Syllabus component | Teaching hours | |
|---|---|---|
| | SL | HL |
| **Syllabus content** | **105** | **195** |
| **Theme A: Concepts of computer science** | | |
| A1 Computer fundamentals | 11 | 18 |
| A2 Networks | 11 | 18 |
| A3 Databases | 11 | 18 |
| A4 Machine learning | 5 | 18 |
| **Theme B: Computational thinking and problem-solving** | | |
| B1 Computational thinking | 5 | 5 |
| B2 Programming | 40 | 42 |
| B3 Object-oriented programming | 7 | 23 |
| B4 Abstract data types—HL only | – | 23 |
| **Case study** | 15 | 30 |
| **Internal assessment**<br>The computational solution | **35**<br>35 | **35**<br>35 |
| **Collaborative sciences project** | **10** | **10** |
| **Total teaching hours** | **150** | **240** |

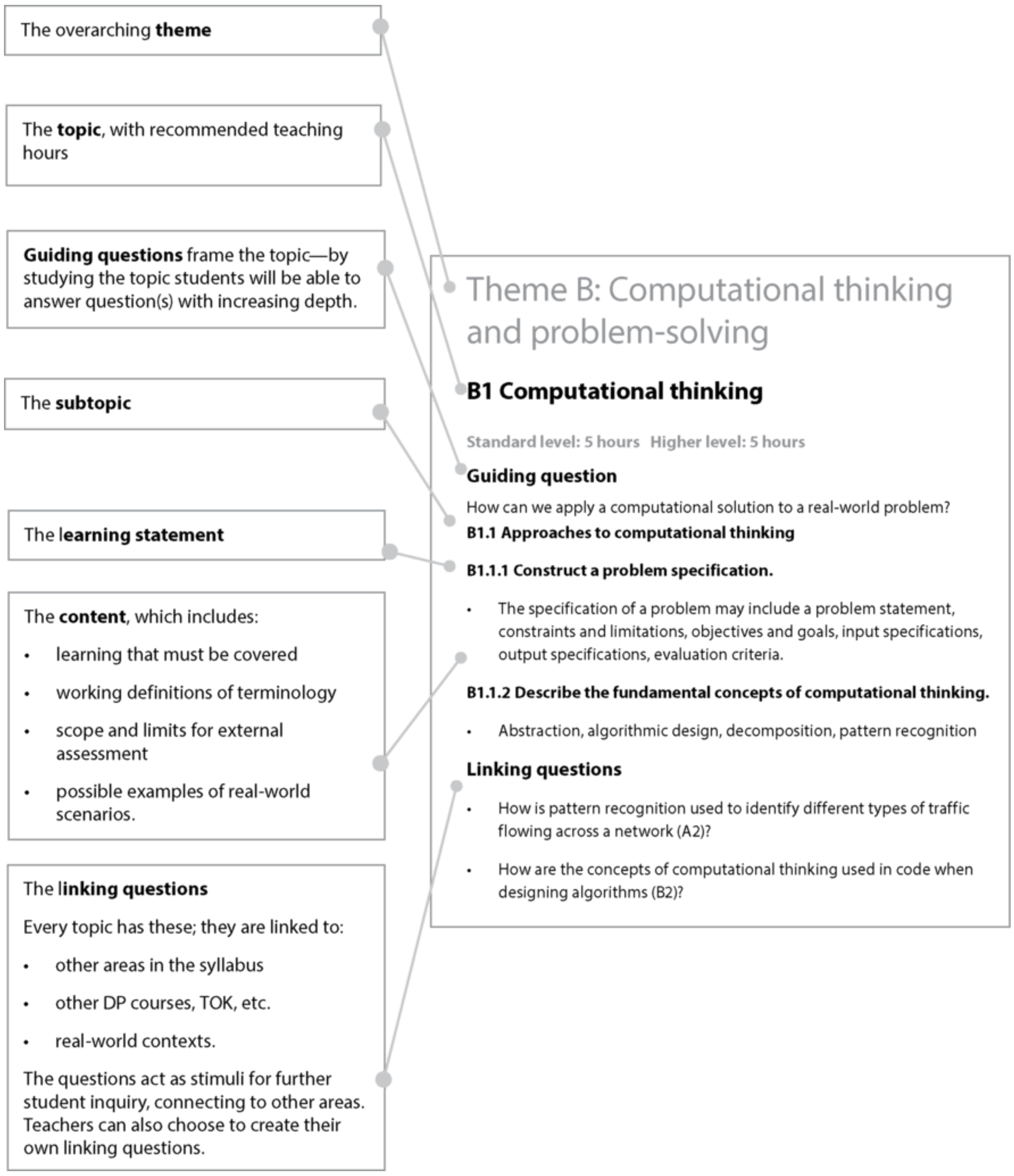
The recommended teaching time is 240 hours to complete HL courses and 150 hours to complete SL courses, as stated in section "B1 General regulations: Diploma Programme" of the publication *Diploma Programme Assessment procedures* (updated annually).

# Syllabus roadmap

The aim of the syllabus is to integrate concepts, topic content and the skills of computer science through inquiry. Students and teachers are encouraged to personalize their approach to the syllabus according to their circumstances and interests.

| Theme A: Concepts of computer science | Theme B: Computational thinking and problem-solving |
|---|---|
| **A1 Computer fundamentals**<br>• A1.1 Computer hardware and operation<br>• A1.2 Data representation and computer logic<br>• A1.3 Operating systems and control systems<br>• A1.4 Translation (HL only) | **B1 Computational thinking**<br>B1.1 Approaches to computational thinking |
| **A2 Networks**<br>• A2.1 Network fundamentals<br>• A2.2 Network architecture<br>• A2.3 Data transmissions<br>• A2.4 Network security | **B2 Programming**<br>• B2.1 Programming fundamentals<br>• B2.2 Data structures<br>• B2.3 Programming constructs<br>• B2.4 Programming algorithms<br>• B2.5 File processing |
| **A3 Databases**<br>• A3.1 Database fundamentals<br>• A3.2 Database design<br>• A3.3 Database programming<br>• A3.4 Alternative databases and data warehouses (HL only) | **B3 Object-oriented programming**<br>• B3.1 Fundamentals of OOP for a single class<br>• B3.2 Fundamentals of OOP for multiple classes (HL only) |
| **A4 Machine learning**<br>• A4.1 Machine learning fundamentals<br>• A4.2 Data preprocessing (HL only)<br>• A4.3 Machine learning approaches (HL only)<br>• A4.4 Ethical considerations | **B4 Abstract data types—HL only**<br>B4.1 Fundamentals of ADTs |

# Syllabus format



The overarching **theme**

The **topic**, with recommended teaching hours

**Guiding questions** frame the topic—by studying the topic students will be able to answer question(s) with increasing depth.

The **subtopic**

The **learning statement**

The **content**, which includes:

- learning that must be covered
- working definitions of terminology
- scope and limits for external assessment
- possible examples of real-world scenarios.

The **linking questions**

Every topic has these; they are linked to:

- other areas in the syllabus
- other DP courses, TOK, etc.
- real-world contexts.

The questions act as stimuli for further student inquiry, connecting to other areas. Teachers can also choose to create their own linking questions.

Theme B: Computational thinking and problem-solving

**B1 Computational thinking**

Standard level: 5 hours   Higher level: 5 hours

**Guiding question**

How can we apply a computational solution to a real-world problem?

**B1.1 Approaches to computational thinking**

**B1.1.1 Construct a problem specification.**

- The specification of a problem may include a problem statement, constraints and limitations, objectives and goals, input specifications, output specifications, evaluation criteria.

**B1.1.2 Describe the fundamental concepts of computational thinking.**

- Abstraction, algorithmic design, decomposition, pattern recognition

**Linking questions**

- How is pattern recognition used to identify different types of traffic flowing across a network (A2)?
- How are the concepts of computational thinking used in code when designing algorithms (B2)?

# Syllabus content

**A note on syllabus scope and limits**

The content of each topic below is sometimes clarified by the terms “may include” and “must include”:

—“**May include**” means that the list includes examples that could be assessed, although other examples may also be used by the IB in assessment.

— “**Must include**” means that the list includes **all** the examples that will be used in assessment.

## Theme A: Concepts of computer science

### A1 Computer fundamentals

Standard level: 11 hours    Higher level: 18 hours

#### Guiding question

What principles underpin the operation of a computer, from low-level hardware functionality to operating system interactions?

#### A1.1 Computer hardware and operation

**A1.1.1 Describe the functions and interactions of the main CPU components.**

- Units: arithmetic logic unit (ALU), control unit (CU)
- Registers: instruction register (IR), program counter (PC), memory address register (MAR), memory data register (MDR), accumulator (AC)
- Buses: address, data, control
- Processors: single core processor, multi-core processor, co-processors
- A diagrammatic representation of the relationship between the specified CPU components

**A1.1.2 Describe the role of a GPU.**

- The architecture that allows graphics processing units (GPUs) to handle specific tasks and makes them suitable for complex computations
- Real-world scenarios may include video games, artificial intelligence (AI), large simulations and other applications that require graphics rendering and machine learning.

**A1.1.3 Explain the differences between the CPU and the GPU. (HL only)**

- Differences in their design philosophies, usage scenarios
- Differences in their core architecture, processing power, memory access, power efficiency
- CPUs and GPUs working together: task division, data sharing, coordinating execution

**A1.1.4 Explain the purposes of different types of primary memory.**

- Random-access memory (RAM), read only memory (ROM), cache (L1, L2, L3), registers
- The interaction of the CPU with different types of memory to optimize performance
- The relevance of the terms “cache miss” and “cache hit”

**A1.1.5 Describe the fetch, decode and execute cycle.**

- The basic operations a CPU performs to execute a single instruction in machine language
- The interaction between memory and registers via the three buses: address, data, control

**A1.1.6 Describe the process of pipelining in multi-core architectures. (HL only)**

- The instructions fetch, decode, execute
- Write-back stages to improve the overall system performance in multi-core architectures
- Overview of how cores in multi-core processors work independently and in parallel

**A1.1.7 Describe internal and external types of secondary memory storage.**

- Internal hard drives: solid state drive (SSD), hard disk drive (HDD), embedded multimedia cards (eMMCs)
- External hard drives: SSD, HDD, optical drives, flash drives, memory cards, network attached storage (NAS)
- The scenarios in which the various types of drive are used

**A1.1.8 Describe the concept of compression.**

- The differences between lossy compression methods and lossless compression methods
- Run-length encoding and transform coding

**A1.1.9 Describe the different types of services in cloud computing.**

- Services: software as a service (SaaS), platform as a service (PaaS), infrastructure as a service (IaaS)
- The differences between the approaches of SaaS, PaaS, and IaaS in various real-world scenarios, recognizing that different degrees of control and flexibility influence resource management and resource availability

## A1.2 Data representation and computer logic

**A1.2.1 Describe the principal methods of representing data.**

- The representation of integers in binary and hexadecimal
- Conversion of binary and hexadecimal integers to decimal, and vice versa
- Conversion of integers from binary to hexadecimal, and vice versa

**A1.2.2 Explain how binary is used to store data.**

- The fundamentals of binary encoding and the impact on data storage and retrieval
- The mechanisms by which data such as integers, strings, characters, images, audio and video are stored in binary form

**A1.2.3 Describe the purpose and use of logic gates.**

- The purpose and use of logic gates
- The functions and applications of logic gates in computer systems
- The role of logic gates in binary computing
- Boolean operators: AND, OR, NOT, NAND, NOR, XOR, XNOR

**A1.2.4 Construct and analyse truth tables.**

- Truth tables to predict the output of simple logic circuits
- Truth tables to determine outputs from inputs for a problem description
- Truth tables and their relationship to a Boolean expression, with inputs and outputs
- Truth tables derived from logic diagrams to aid the simplification of logical expressions
- Karnaugh maps and algebraic simplification to simplify output expressions

**A.1.2.5 Construct logic diagrams.**

- Logic diagrams to demonstrate how logic gates are connected and interact in a circuit.
- Use of standard gate symbols for AND, OR, NOT, NAND, NOR, XOR and XNOR gates
- Inputs processed diagrammatically to produce outputs
- Combinations of these gates to perform more complex logical operations
- Boolean algebra rules to simplify complex logic diagrams and expressions

### A1.3 Operating systems and control systems

**A1.3.1 Describe the role of operating systems.**

- Operating systems abstract hardware complexities to manage system resources

**A1.3.2 Describe the functions of an operating system.**

- Maintaining system integrity while running operating systems' background operations
- Memory management, file system, device management, scheduling, security, accounting, graphical user interface (GUI), virtualization, networking

**A1.3.3 Compare different approaches to scheduling.**

- Managing the execution of processes by allocating CPU time to optimize system performance
- First-come first-served, round robin, multilevel queue scheduling, priority scheduling

**A1.3.4 Evaluate the use of polling and interrupt handling.**

- Event frequency, CPU processing overheads, power source (battery or mains), event predictability, controlled latency, security concerns
- Real-world scenarios may include keyboard and mouse inputs, network communications, disk input/output operations, embedded systems, real-time systems.

**A1.3.5 Explain the role of the operating system in managing multitasking and resource allocation. (HL only)**

- The challenges of multitasking and resource allocation, including task scheduling, resource contention and deadlock

**A1.3.6 Describe the use of the control system components. (HL only)**

- The input, process, output, and feedback mechanism (open-loop, closed-loop)
- Controller, sensors, actuators, transducers and control algorithm

**A1.3.7 Explain the use of control systems in a range of real-world applications. (HL only)**

- Examples may include autonomous vehicles, home thermostats, automatic elevator controllers, automatic washing machines, traffic signal control systems, irrigation control systems, home security systems, automatic doors.

### A1.4 Translation (HL only)

**A1.4.1 Evaluate the translation processes of interpreters and compilers.**

- The mechanics and use-cases of each translation approach
- The difference in error detection, translation time, portability and applicability for different translation processes, including just-in-time compilation (JIT) and bytecode interpreters
- Example scenarios where the translation method should be considered must include rapid development and testing, performance-critical applications and cross-platform development.

### Linking questions

- What role does multitasking in an operating system play in machine learning (A4)?
- How might a conditional statement be constructed using Boolean logic gates in a circuit (B2)?
- What role does task-scheduling in an operating system play in managing network traffic and requests (A2.3.3, A2.4)?
- How does resource allocation in an operating system impact network performance and stability (A2)?
- What role do GPUs play in non-graphics computational tasks (A4)?
- To what extent should computer systems not cause harm (TOK)?

## A2 Networks

Standard level: 11 hours    Higher level: 18 hours

## Guiding question

What are the principles and concepts that underpin how networks operate?

## A2.1 Network fundamentals

**A2.1.1 Describe the purpose and characteristics of networks.**

- Networks: local area network (LAN), wide area network (WAN), personal area network (PAN), virtual private network (VPN)

**A2.1.2 Describe the purpose, benefits and limitations of modern digital infrastructures.**

- Modern digital infrastructure: the internet, cloud computing, distributed systems, edge computing, mobile networks
- Examples where specific networks are used may include the worldwide web (WWW), cryptocurrency blockchains, smart traffic lights, a school.

**A2.1.3 Describe the function of network devices.**

- Gateways, hardware firewalls, modems, network interface cards, routers, switches, wireless access points
- How devices map to the layers of the TCP/IP model

**A2.1.4 Describe the network protocols used for transport and application.**

- Protocols: transmission control protocol (TCP), user datagram protocol (UDP), hypertext transfer protocol (HTTP), hypertext transfer protocol secure (HTTPS), dynamic host configuration protocol (DHCP)

**A2.1.5 Describe the function of the TCP/IP model. (HL only)**

- Application, transport, internet, network interface
- The role of each layer and the interaction between these layers to ensure reliable data transmission over a network

## A2.2 Network architecture

**A2.2.1 Describe the functions and practical applications of network topologies.**

- Network topologies: star, mesh, hybrid
- Factors to consider must include reliability, transmission speed, scalability, data collisions, cost.
- Examples may include home and small office settings, where reliability is paramount, and the use of networks in larger settings (e.g. corporations, government departments, college campuses).

**A2.2.2 Describe the function of servers. (HL only)**

- Types of servers: domain name server (DNS), dynamic host configuration protocol (DHCP), file server, mail server, proxy server, web server
- Factors to consider must include function, scalability, reliability and security.

**A2.2.3 Compare and contrast networking models.**

- Client-server and peer-to-peer models
- The respective benefits and drawbacks of client-server and peer-to-peer models
- Real-world applications may include web browsing, email services, online banking, file sharing, VoIP services, blockchain.

**A2.2.4 Explain the concepts and applications of network segmentation.**

- Segmentation for network performance and security, to reduce congestion, to manage network resources efficiently
- Network segmentation must include the uses and roles of segmenting, subnetting and virtual local area networks (VLANs).

## A2.3 Data transmissions

**A2.3.1 Describe different types of IP addressing.**

- The distinction between IPv4 and IPv6 addressing
- The differences between public IP addresses and private IP addresses, and between static IP addresses and dynamic IP addresses
- The role of network address translation (NAT) to minimize the use of IP addresses and to facilitate communication between private internal networks and the public internet

**A2.3.2 Compare types of media for data transmission.**

- Wired transmission via fibre optic cables and twisted pair cables; wireless transmission
- The advantages and disadvantages of these three types of data transmission
- Factors to consider must include bandwidth, complexity of installation, cost, range, susceptibility to interference, attenuation, reliability, security.

**A2.3.3 Explain how packet switching is used to send data across a network.**

- The process of segmenting data into packets with a routing header attached, and independently transmitting control information, allowing the data to be reassembled at the destination
- The role that switches and routers play in packet switching

**A2.3.4 Explain how static routing and dynamic routing move data across local area networks. (HL only)**

- The process of static routing, and its advantages and disadvantages
- The process of dynamic routing, and its advantages and disadvantages (explanation of a specific routing protocol is not required)
- Factors to consider must include configuration, maintenance, complexity, resource usage, convergence, scalability, network size.

## A2.4 Network security

**A2.4.1 Discuss the effectiveness of firewalls at protecting a network.**

- The function of firewalls in inspecting and filtering incoming and outgoing traffic based on whitelists, blacklists and rules
- The strengths and limitations of firewalls
- The role of NAT to enhance network security

**A2.4.2 Describe common network vulnerabilities. (HL only)**

- Distributed denial of service (DDoS), insecure network protocols, malware, man-in-the-middle (MitM) attacks, phishing attacks, SQL injection, cross-site scripting (XSS), unpatched software, weak authentication, zero-day exploits

**A2.4.3 Describe common network countermeasures. (HL only)**

- Content security policies, complex password policies, DDoS mitigation tools, email filtering solutions, encrypted protocols, input validation (filtering, whitelisting), intrusion detection systems (IDS), intrusion prevention systems (IPS), multifactor authentication (MFA), secure socket layer (SSL) certificate, transport layer security (TLS) certificate, update software, VPNs
- The importance of regular security testing and employee training
- Wireless security measures may include media access controllers (MAC), whitelists and blacklists.

**A2.4.4 Describe the process of encryption and digital certificates.**

- The difference between symmetric and asymmetric cryptography
- The role of digital certificates in establishing secure network connections
- The use of public and private keys in asymmetric cryptography
- The significance of encryption key management

## Linking questions

- Do networks and databases use the same form of encryption algorithms (A3)?

- How do cloud computing and distributed systems utilize networking to deliver services (A1.1.9)?
- How do the concepts of binary and hexadecimal data structures relate to network communications (B2)?
- Are similar ethical principles needed when transmitting data over a network and using data in machine learning algorithms (TOK)?
- How can network types or transmissions impact database performance (A3)?
- What are the similarities and differences between network security and database security (A3)?
- How do network technologies influence machine learning algorithms (A4)?

## A3 Databases

Standard level: 11 hours    Higher level: 18 hours

### Guiding question

What are the principles, structures, and operations that form the basis of database systems?

### A3.1 Database fundamentals

**A3.1.1 Explain the features, benefits and limitations of a relational database.**

- Features: composite keys, foreign keys, primary keys, relationships, tables
- Benefits of databases: community support, concurrency control, data consistency, data integrity, data retrieval, reduced data duplication, reduced redundancy, reliable transaction processing, scalability, security features
- Limitations of databases: “big data” scalability issues, design complexity, hierarchical data handling, rigid schema, object-relational impedance mismatch, unstructured data handling

### A3.2 Database design

**A3.2.1 Describe database schemas.**

- Conceptual schema, logical schema, physical schema
- Abstract definitions of the data structure and organization of the data at different levels

**A3.2.2 Construct ERDs.**

- The significance of entity relationship diagrams (ERDs) in crafting organized, efficient database designs tailored for specific applications
- The relationships between different data entities within a database
- The roles of cardinality and modality in defining relationships in ERDs

**A3.2.3 Outline the different data types used in relational databases.**

- The importance of data type consistency
- The potential effects of choosing the wrong data type

**A3.2.4 Construct tables for relational databases.**

- The relationship between tables using primary keys, foreign keys, composite keys and concatenated keys
- The importance of well-defined tables in ensuring data integrity

**A3.2.5 Explain the difference between normal forms.**

- First normal form (1NF), second normal form (2NF), third normal form (3NF)
- The terms atomicity, unique identification, functional dependencies, partial-key dependencies, non-key/transitive dependencies
- Normalization issues can encompass data duplication, missing data, and a range of dependency concerns, including data dependencies, composite key dependencies, transitive dependencies, and multi-valued dependencies.

**A3.2.6 Construct a database normalized to 3NF for a range of real-world scenarios.**

- Examples may include library management, hospital management, e-commerce platforms, school management, employee management, inventory management, police crime reporting

**A3.2.7 Evaluate the need for denormalizing databases.**

- The advantages and disadvantages of normalizing and denormalizing databases
- Situations where denormalization can enhance performance, particularly in read-intensive applications
- The balance between straightforward query structures and the risk of data redundancy in denormalized schemas

## A3.3 Database programming

**A3.3.1 Outline the differences between data language types within SQL.**

- Data language types must include data definition language (DDL) and data manipulation language (DML)
- SQL statements to define data structures or to manipulate data

**A3.3.2 Construct queries between two tables in SQL.**

- Queries must include joins, relational operators, filtering, pattern matching, and ordering data
- SQL commands: SELECT, DISTINCT, FROM, WHERE, BETWEEN, ORDER BY, GROUP BY, HAVING, ASC, DESC, JOIN, LIKE with % wildcard, AND, OR, NOT (**note:** Syntax may vary in different database systems)

**A3.3.3 Explain how SQL can be used to update data in a database.**

- Insert new records (INSERT INTO), modify data (UPDATE SET), remove data (DELETE)
- The performance implications of updating data in indexed columns, and how indexes might need to be rebuilt or reorganized following significant data modifications

**A3.3.4 Construct calculations within a database using SQL's aggregate functions. (HL only)**

- Aggregate functions on grouped data to aid reporting and decision-making
- Aggregate commands: AVERAGE, COUNT, MAX, MIN, SUM

**A3.3.5 Describe different database views. (HL only)**

- Virtual views and materialized (snapshot) views
- Hiding data complexity, data consistency, independence, performance, query simplification, read-only data or updatable data, security

**A3.3.6 Describe how transactions maintain data integrity in a database. (HL only)**

- The role of atomicity, consistency, isolation and durability (ACID) to ensure reliable processing of transactions
- Transaction control language (TCL) commands: BEGIN TRANSACTION, COMMIT, ROLLBACK

## A3.4 Alternative databases and data warehouses (HL only)

**A3.4.1 Outline the different types of databases as approaches to storing data.**

- Databases models: NoSQL, cloud, spatial, in-memory
- Examples of the use of the database model in real-world scenarios may include e-commerce platforms, geographic information systems (GIS), managed services, real-time analytics, social media platforms, SaaS.

**A3.4.2 Explain the primary objectives of data warehouses in data management and business intelligence.**

- The roles of append-only data, subject-oriented data, integrated data, time-variant data, non-volatile data and data optimized for query performance, to ensure efficient data storage and analysis

**A3.4.3 Explain the role of online analytical processing (OLAP) and data mining for business intelligence.**

- Data mining techniques must include classification, clustering, regression, association rule discovery, sequential pattern discovery, anomaly detection (**note**: This links to "A4 Machine learning").
- The uses of the techniques in extracting meaningful information from large data sets

**A3.4.4 Describe the features of distributed databases.**

- The need to maintain data consistency in a distributed database
- The role of ACID to ensure reliable processing of transactions in distributed databases
- Features of distributed databases: concurrency control, data consistency, data partitioning, data security, distribution transparency, fault tolerance, global query processing, location transparency, replication, scalability

### Linking questions

- What processes are needed to store data in database structures so that they can be used in machine learning (A4)?
- How does database programming in SQL differ from programming computationally in a high-level language (B2)?
- To what extent is the effectiveness of the distributed database determined by the network that connects the various tables (A2)?
- How could machine learning be applied to databases (A4)?
- How do programming languages interact with databases to store, retrieve and manipulate data (B2)?

## A4 Machine learning

Standard level: 5 hours    Higher level: 18 hours

### Guiding question

What principles and approaches should be considered to ensure machine learning models produce accurate results ethically?

### A4.1 Machine learning fundamentals

**A4.1.1 Describe the types of machine learning and their applications in the real world.**

- The different approaches to machine learning algorithms and their unique characteristics
- Deep learning (DL), reinforcement learning (RL), supervised learning, transfer learning (TL), unsupervised learning (UL)
- Real-world applications of machine learning may include market basket analysis, medical imaging diagnostics, natural language processing, object detection and classification, robotics navigation, sentiment analysis.

**A4.1.2 Describe the hardware requirements for various scenarios where machine learning is deployed.**

- The hardware configurations for different machine learning scenarios, considering factors such as processing, storage and scalability
- Hardware configurations for machine learning ranging from standard laptops to advanced infrastructure
- Advanced infrastructure must include application-specific integrated circuits (ASICs), edge devices, field-programmable gate arrays (FPGAs), GPUs, tensor processing units (TPUs), cloud-based platforms, high-performance computing (HPC) centres.

### A4.2 Data preprocessing (HL only)

**A4.2.1 Describe the significance of data cleaning.**

- The impact of data quality on model performance

- Techniques for handling outliers, removing or consolidating duplicate data, identifying incorrect data, filtering irrelevant data, transforming improperly formatted data, and imputation, deletion or predictive modelling for missing data
- Normalization and standardization as crucial preprocessing steps

**A4.2.2 Describe the role of feature selection.**

- Feature selection to identify and retain the most informative attributes of the data set
- Feature selection strategies: filter methods, wrapper methods, embedded methods

**A4.2.3 Describe the importance of dimensionality reduction.**

- The curse of dimensionality considerations may include overfitting, computational complexity, data sparsity, the effectiveness of distance metrics, data visualization, sample size increases, memory usage.
- Dimensionality reduction of variables, while preserving the relevant aspects of the data

**Note:** Statistical techniques such as principal component analysis (PCA) and linear discriminant analysis (LDA) are beyond the scope of this course.

## A4.3 Machine learning approaches (HL only)

**A4.3.1 Explain how linear regression is used to predict continuous outcomes.**

- The relationship between the independent (predictor) and dependent (response) variables
- The significance of the slope and intercept in the regression equation
- How well the model fits the data—often assessed using measures like $r^2$.

**A4.3.2 Explain how classifications techniques in supervised learning are used to predict discrete categorical outcomes.**

- K-Nearest Neighbours (K-NN) and decision trees algorithms to categorize new data points, based on patterns learned from existing labelled data
- Real-world applications of K-NN may include collaborative filtering recommendation systems.
- Real-world applications of decision trees may include medical diagnosis based on a patient's symptoms.

**A4.3.3 Explain the role of hyperparameter tuning when evaluating supervised learning algorithms.**

- Accuracy, precision, recall and F1 score as evaluation metrics
- The role of hyperparameter tuning on model performance
- Overfitting and underfitting when training algorithms

**A4.3.4 Describe how clustering techniques in unsupervised learning are used to group data based on similarities in features.**

- Clustering techniques in unsupervised learning group data based on feature similarities
- Real-world applications of clustering may include using purchasing data to segment a customer base.

**A4.3.5 Describe how learning techniques using the association rule are used to uncover relations between different attributes in large data sets.**

- Mining techniques using the association rule and interpretation of the results for a given scenario

For example, in crime analysis, the techniques may reveal that areas with high rates of vandalism also often experience incidents of theft, assisting law enforcement in predictive policing and resource allocation.

**A4.3.6 Describe how an agent learns to make decisions by interacting with its environment in reinforcement learning.**

- The principle of cumulative reward and the foundational concepts of agent–environment interaction, encompassing actions, states, rewards and policies
- The exploration versus exploitation trade-off as a core concept in reinforcement learning

**A4.3.7 Describe the application of genetic algorithms in various real-world situations.**

- For example: population, fitness function, selection, crossover, mutation, evaluation, termination

- A real-world application of genetic algorithms is seen in optimization problems, such as route planning (e.g. the "travelling salesperson problem").

**A4.3.8 Outline the structure and function of ANNs and how multi-layer networks are used to model complex patterns in data sets.**

- An artificial neural network (ANN) to simulate interconnected nodes or "neurons" to process and learn from input data, enabling tasks such as classification, regression and pattern recognition
- Sketch of a single perceptron, highlighting its input, weights, bias, activation function and output
- Sketch of a multi-layer perceptron (MLP) encompassing the input layer, one or more hidden layers and the output layer.

**A4.3.9 Describe how CNNs are designed to adaptively learn spatial hierarchies of features in images.**

- Convolutional neural network (CNN) basic architecture: input layer, convolutional layers, activation functions, pooling layers, fully connected layers, output layer
- The effect of the number of layers, kernel size and stride, activation function selection, and the loss function on how CNNs process input data and classify images

**A4.3.10 Explain the importance of model selection and comparison in machine learning.**

- How different algorithms can yield different results depending on the data and type of problem
- The reasons for selecting specific machine learning models over others, considering factors like the nature of the problem, its complexity and desired outcomes
- The variability in algorithm performance based on the data's characteristics

### A4.4 Ethical considerations

**A4.4.1 Discuss the ethical implications of machine learning in real-world scenarios.**

- Ethical issues may include accountability, algorithmic fairness, bias, consent, environmental impact, privacy, security, societal impact, transparency.
- The challenges posed by biases in training data
- The ethics of using machine learning in online communication may include concerns about misinformation, bias, online harassment, anonymity, privacy.

**A4.4.2 Discuss ethical aspects of the increasing integration of computer technologies into daily life.**

- The importance of continually reassessing ethical guidelines as technology advances
- The potential implications of emerging technologies such as quantum computing, augmented reality, virtual reality and the pervasive use of AI on society, individual rights, privacy and equity

### Linking questions

- How can machine learning be applied to optimize the management of network traffic (A2)?
- How does database programming in SQL differ from programming computationally in a high-level language (A3, B2)?
- To what extent are developments in machine learning ethical (TOK)?
- How can larger models be processed using GPUs and cloud processing (A1)?
- Can machine learning find and improve network security problems (A2)?

# Theme B: Computational thinking and problem-solving

## B1 Computational thinking

Standard level: 5 hours Higher level: 5 hours

### Guiding question

How can we apply a computational solution to a real-world problem?

### B1.1 Approaches to computational thinking

**B1.1.1 Construct a problem specification.**

- The specification of a problem may include a problem statement, constraints and limitations, objectives and goals, input specifications, output specifications, evaluation criteria.

**B1.1.2 Describe the fundamental concepts of computational thinking.**

- Abstraction, algorithmic design, decomposition, pattern recognition

**B1.1.3 Explain how applying computational thinking to fundamental concepts is used to approach and solve problems in computer science.**

- Computational thinking does not necessarily involve programming—it is a toolkit of available techniques for problem-solving.
- Real-world examples may include software development, data analysis, machine learning, database design, network security.

**B1.1.4 Trace flowcharts for a range of programming algorithms.**

- Use of standard flowchart symbols to depict processes, decisions and flows of control
- Standard flowchart symbols: Connector, Decision, Flowline, Input/Output, Process/Operation, Start/End
- Flowcharts for execution flow, to track changes in variables and to determine output

### Linking questions

- How is pattern recognition used to identify different types of traffic flowing across a network (A2)?
- How are the concepts of computational thinking used in code when designing algorithms (B2)?

## B2 Programming

Standard level: 40 hours    Higher level: 42 hours

### Guiding question

How can we apply computer programming to solve problems?

### B2.1 Programming fundamentals

**B2.1.1 Construct and trace programs using a range of global and local variables of various data types.**

- Data types: Boolean value, char, decimal, integer, string

**B2.1.2 Construct programs that can extract and manipulate substrings.**

- Writing of programs that accurately identify and extract substrings from given strings, demonstrating the ability to perform various manipulations, such as altering, concatenating or replacing

**B2.1.3 Describe how programs use common exception handling techniques.**

- Potential points of failure in a program must include unexpected inputs, resource unavailability, logic errors.
- The role of exception handling in developing programs
- Exception handling constructs that effectively manage errors must include try/catch in Java, and try/except in Python, along with the finally block.

**B2.1.4 Construct and use common debugging techniques.**

- Debugging techniques may include trace tables, breakpoint debugging, print statements and step-by-step code execution.

## B2.2 Data structures

**B2.2.1 Compare static and dynamic data structures.**

- The fundamental differences between static and dynamic data structures, including their underlying mechanisms for memory allocation and resizing
- The advantages and disadvantages of each type in various scenarios, considering factors such as speed, memory usage, flexibility

**B2.2.2 Construct programs that apply arrays and Lists.**

- One-dimensional (1D) arrays, two-dimensional (2D) arrays, ArrayLists in Java
- One-dimensional (1D) Lists and two-dimensional (2D) Lists in Python
- Add, remove and traverse elements in a dynamic list

**B2.2.3 Explain the concept of a stack as a "last in, first out" (LIFO) data structure.**

- Must include fundamental operations such as push, pop, peek and isEmpty
- How stack operations impact both performance and memory usage
- An appropriate stack for a specific problem

**B2.2.4 Explain the concept of a queue as a "first in, first out" (FIFO) data structure.**

- Must include fundamental operations such as enqueue, dequeue, front and isEmpty
- How queue operations impact both performance and memory usage
- An appropriate queue for a specific problem

## B2.3 Programming constructs

**B2.3.1 Construct programs that implement the correct sequence of code instructions to meet program objectives.**

- The impact of instruction order on program functionality
- Ways to avoid errors, such as infinite loops, deadlock, incorrect output

**B2.3.2 Construct programs utilizing appropriate selection structures.**

- Must include: if, else, else if (Java), elif (Python), to execute different code blocks based on specified conditions
- Selection structures with or without Boolean operators (AND, OR, NOT) and/or relational operators (<, <=, >, >=, ==, !=) to control program flow effectively

**B2.3.3 Construct programs that utilize looping structures to perform repeated actions.**

- Types of loops, including counted loops and conditional loops, and appropriate use of each type
- Conditional statements within loops, using Boolean and/or relational operators to govern the loop's execution

**B2.3.4 Construct functions and modularization.**

- Functions to define reusable blocks of code with different inputs
- Modularization to create well-structured, reusable and maintainable code
- The principles of scope (local versus global)
- The benefits of code modularization, applying this concept to various programming scenarios

## B2.4 Programming algorithms

**B2.4.1 Describe the efficiency of specific algorithms by calculating their Big O notation to analyse their scalability.**

- The time and space complexities of algorithms and calculating Big O notation
- Algorithm choice based on scalability and efficiency requirements

**B2.4.2 Construct and trace algorithms to implement a linear search and a binary search for data retrieval.**

- The differences in efficiency between different methods of linear and binary search
- Use of search technique based on efficiency requirements—for example, searching a database for a sorted/indexed list of names to find a phone number, versus searching by the number to identify the name

**B2.4.3 Construct and trace algorithms to implement bubble sort and selection sort, evaluating their time and space complexities.**

- The time and space complexities of each algorithm, denoted by their respective Big O notations
- The advantages and disadvantages of each algorithm in terms of efficiency across various data sets

**B2.4.4 Explain the fundamental concept of recursion and its applications in programming. (HL only)**

- The fundamentals of recursion and its advantages and limitations
- The utility of recursion in solving problems that can be broken down into smaller, similar sub-problems
- Recursive algorithms, including but not limited to quicksort
- The limitations of recursion, including complexity and memory usage
- Situations that best suit the use of recursion, including fractal image creation, traversing binary trees, sorting algorithms

**B2.4.5 Construct and trace recursive algorithms in a programming language. (HL only)**

- Simple, non-branching recursive algorithms in programming only

### B2.5 File processing

**B2.5.1 Construct code to perform file-processing operations.**

- Programs that manipulate text files
- Opening a sequential file in various modes (read, write, append)
- How to read from and write to files, append data to an existing file, and close a file once operations are completed
- Classes for Java users may include Scanner, FileWriter, BufferedReader.
- Functions for Python users may include open(), read(), readline(), write(), close().

### Linking questions

- Does database programming in SQL require computational thinking (A3)?
- Why is an understanding of variables and their scope important for effective memory management in computer systems (A1)?
- Is algorithmic efficiency relevant to machine learning, where large data sets are processed and computational cost can be significant (A4)?
- Are data structures like stacks and queues applicable in networking algorithms for packet routing and load balancing (A2)?
- How can graph theory be applied to packet distribution in networks (A2, DP mathematics: applications and interpretation—HL)?
- How do graph algorithms and terminologies, such as vertices and edges, impact machine learning algorithms like network analysis (A4, DP mathematics: applications and interpretation—HL)?
- How can network traffic be used as an example of, or connection to, programming algorithms (A2)?
- How could programming algorithms be applied to develop machine learning methods (A4)?

## B3 Object-oriented programming

Standard level: 7 hours Higher level: 23 hours

### Guiding question

Is object-oriented programming (OOP) an appropriate paradigm for solving complex problems?

## B3.1 Fundamentals of OOP for a single class

**B3.1.1 Evaluate the fundamentals of OOP.**

- Model real-world entities using OOP concepts: classes, objects, inheritance, encapsulation, polymorphism
- The advantages and disadvantages of using OOP in various programming scenarios

**B3.1.2 Construct a design of classes, their methods and behaviour.**

- Classes and their methods, based on application requirements
- The use of unified modelling language (UML) class diagrams to represent class relationships, attributes and methods, to aid effective software design and planning

**B3.1.3 Distinguish between static and non-static variables and methods.**

- The differences between static and non-static variables and methods, including their usage and scope
- When to use instance variables instead of class variables, and how to apply these concepts effectively in code

**B3.1.4 Construct code to define classes and instantiate objects.**

- How to define classes and create objects from those classes
- The role of constructors in initializing an object's state, setting initial values for its attributes to define its condition or characteristics at the time of creation

**B3.1.5 Explain and apply the concepts of encapsulation and information hiding in OOP.**

- The principles of encapsulation and information hiding
- Apply access modifiers such as private and public
- Controlling access to class members
- The importance of limiting access to maintain the integrity and security of an object's state

## B3.2 Fundamentals of OOP for multiple classes (HL only)

**B3.2.1 Explain and apply the concept of inheritance in OOP to promote code reusability.**

- How inheritance enables a hierarchical relationship between parent and child classes
- Extending existing classes, utilizing inheritance to reuse and extend functionalities
- The impact of inheritance on access to parent class members with different access modifiers (private, public, protected, default)

**B3.2.2 Construct code to model polymorphism and its various forms, such as method overriding.**

- The principle of polymorphism and how it contributes to code flexibility and reusability
- How to implement dynamic polymorphic behaviour through mechanisms like method overriding
- How to apply static polymorphic behaviour to maximize code efficiency

**B3.2.3 Explain the concept of abstraction in OOP.**

- The significance of abstraction in the development of modular code fragments
- The use of abstract classes to establish common interfaces for sub-classes

**B3.2.4 Explain the role of composition and aggregation in class relationships.**

- How to design objects by leveraging smaller component objects through composition and aggregation
- That aggregation implies that the subcomponents can function independently of the aggregating class, while in composition, the subcomponents are tightly coupled and cannot exist outside the aggregating class

**B3.2.5 Explain commonly used design patterns in OOP.**

- The key design patterns such as singleton, factory and observer
- The application of design patterns in solving recurring programming challenges

### Linking questions

- In what ways can OOP be applied to database development (A3)?
- Is OOP necessary for all programming, or only in modelling complex situations (B2)?
- How can design patterns in OOP facilitate the architecture of scalable and maintainable machine learning models (A4)?
- How can the principles of encapsulation and information hiding be applied to secure network communication (A3)?

## B4 Abstract data types (HL only)

Higher level: 23 hours

### Guiding question

Which abstract data types (ADTs) are most appropriate for different situations?

### B4.1 Fundamentals of ADTs

**B4.1.1 Explain the properties and purpose of ADTs in programming.**

- The core principles of ADTs, including their purpose in providing a high-level description of data structures and their associated operations

**B4.1.2 Evaluate linked lists.**

- Lists must include singly, doubly, circular
- Sketch of linked lists and implementation of basic operations diagrammatically, such as insertion, deletion, traversal, search
- The advantages and disadvantages of using linked lists over other data structures like arrays, particularly in terms of memory utilization and performance

**B4.1.3 Construct and apply linked lists: singly, doubly and circular.**

- The basic operations on a linked list, such as insertion, deletion, traversal, search

**B4.1.4 Explain the structures and properties of BSTs.**

- How binary search trees (BSTs) are used for data organization
- Insert, delete, traverse and searching nodes in a BST
- Sketching a BST as a tree diagram

**B4.1.5 Construct and apply sets as an ADT.**

- The fundamental characteristics of sets, including their unordered nature and the uniqueness of elements
- Operations: union, intersection and difference
- Code to check if an element is in a set, to add an element to a set, to remove an element, and to check whether one set is a subset/superset of another set

**B4.1.6 Explain the core principles of ADTs.**

- High-level description of data structures and their associated operations and purpose
- The underlying mechanics of hash tables, including hashing functions, collision resolution strategies and load factors
- The underlying mechanics of sets to store and manage data
- HashMap and HashSet in Java; dict and set in Python

### Linking questions

- What role do stacks and queues play in handling CPU interrupts and polling (A1)?
- Can ADTs be used to manage data (A2)?
- How can ADTs be used to optimize file-processing operations like read and write (B2)?

- Can a BST play a role in the quicksort algorithm (B1)?

# Assessment in the Diploma Programme

## General

Assessment is an integral part of learning and teaching. The most important aims of assessment in the DP are that it should support curricular goals and encourage appropriate student learning. Both external and internal assessments are used in the DP. IB examiners mark work produced for external assessment, while work produced for internal assessment is marked by teachers and externally moderated by the IB.

There are two types of assessment identified by the IB.

- Formative assessment informs both learning and teaching. It is concerned with providing accurate and helpful feedback to students and teachers on the kind of learning taking place and the nature of students' strengths and weaknesses in order to help develop students' understanding and capabilities. Formative assessment can also help to improve teaching quality, as it can provide information to monitor progress towards meeting the course aims and objectives.
- Summative assessment gives an overview of previous learning and is concerned with measuring student achievement at, or towards the end of, the course of study.

A comprehensive assessment plan is viewed as being integral with teaching, learning and course organization. For further information, see the IB *Programme standards and practices* publication.

The approach to assessment used by the IB is criterion-related, not norm-referenced. This approach judges students' work by their performance in relation to identified levels of attainment, not in relation to the work of other students. For further information on assessment within the DP please refer to the publication *Assessment principles and practices—Quality assessments in a digital age*.

To support teachers in the planning, delivery and assessment of DP courses, a variety of resources can be found on the Programme Resource Centre or purchased from the IB store (store.ibo.org). Additional publications such as specimen papers and markschemes, teacher support materials, subject reports and grade descriptors can also be found on the Programme Resource Centre. Past examination papers as well as markschemes can be purchased from the IB store.

## Methods of assessment

The IB uses several methods to assess work produced by students.

### Assessment criteria

Assessment criteria are used when the assessment task is open-ended. Each criterion concentrates on a particular skill that students are expected to demonstrate. An assessment objective describes what students should be able to do, and assessment criteria describe how well they should be able to do it. Using assessment criteria allows discrimination between different answers and encourages a variety of responses. Each criterion comprises a set of hierarchically ordered level descriptors. Each level descriptor is worth one or more marks. Each criterion is applied independently using a best-fit model. The maximum marks for each criterion may differ according to the criterion's importance. The marks awarded for each criterion are added together to give the total mark for the piece of work.

### Markbands

Markbands are a comprehensive statement of expected performance against which responses are judged. They represent a single holistic criterion divided into level descriptors. Each level descriptor corresponds to a range of marks to differentiate student performance. A best-fit approach is used to ascertain which particular mark to use from the possible range for each level descriptor.

### Analytic markschemes

Analytic markschemes are prepared for those examination questions that expect a particular kind of response and/or a given final answer from students. They give detailed instructions to examiners on how to break down the total mark for each question for different parts of the response.

### Marking notes

For some assessment components marked using assessment criteria, marking notes are provided. Marking notes give guidance on how to apply assessment criteria to the particular requirements of a question.

## Inclusive access arrangements

Inclusive access arrangements are available for students with access requirements. Standard assessment conditions may put students with assessment access requirements at a disadvantage by preventing them from demonstrating their attainment level. Inclusive access arrangements enable students to demonstrate their ability under assessment conditions that are as fair as possible.

The IB publication *Access and inclusion policy* provides details on all the inclusive access arrangements available to students. The IB publication *Learning diversity and inclusion in IB programmes: Removing barriers to learning* outlines the position of the IB with regard to students with diverse learning needs in the IB programmes. For students affected by adverse circumstances, the publication *Diploma Programme Assessment procedures* (updated annually), which includes the general regulations, provides details on access consideration.

## Responsibilities of the school

The school is required to ensure that equal access arrangements and reasonable adjustments are provided to students with learning support requirements that are in line with the IB publications *Access and inclusion policy* and *Learning diversity and inclusion in IB programmes: Removing barriers to learning*.

# Assessment outline—SL

| First assessment 2027 |
|---|

| Assessment component | Weighting |
|---|---|
| **External assessment (2 hours 30 minutes)** | **70%** |
| **Paper 1 (1 hour 15 minutes)**<br>Section A—extended response questions linked to theme A: Concepts of computer science<br>Section B—short-response questions linked to the pre-seen case study<br>(50 marks) | **35%** |
| **Paper 2 (1 hour 15 minutes)**<br>Extended response questions linked to theme B: Computational thinking and problem-solving<br>(50 marks) | **35%** |
| **Internal assessment (35 hours)**<br>This component is internally assessed by the teacher and externally moderated by the IB at the end of the course.<br>IA consists of one task: the computational solution<br>(30 marks) | **30%** |

# Assessment outline—HL

| First assessment 2027 |
|---|

| Assessment component | Weighting |
|---|---|
| **External assessment (4 hours)** | **80%** |
| **Paper 1 (2 hours)**<br>Section A—extended-response questions linked to theme A: Concepts of computer science<br>Section B—short- and extended-response questions linked to the pre-seen case study<br>(80 marks) | **40%** |
| **Paper 2 (2 hours)**<br>Extended-response questions linked to theme B: Computational thinking and problem-solving<br>(80 marks) | **40%** |
| **Internal assessment (35 hours)**<br>This component is internally assessed by the teacher and externally moderated by the IB at the end of the course.<br>IA consists of one task: the computational solution<br>(30 marks) | **20%** |

# External assessment

Detailed markschemes specific to each examination paper are used for all examinations.

## External assessment details—SL

### Paper 1

**Duration: 1 hour 15 minutes**

**Weighting: 35%**

**Marks: 50**

Paper 1 consists of two sections. All questions on paper 1 are compulsory.

**Section A (38 marks)**

Section A consists of extended-response questions linked to the SL topics in theme A: Concepts of computer science.

**Section B (12 marks)**

Section B consists of short-response questions examining concepts of computer science, based on the pre-seen case study and the two challenge questions given in the case study.

| Section A and section B are completed together without any interruptions. |
|---|

#### Case study

The computer science case study provides the stimulus to investigate a scenario involving current developments, emerging technologies and/or ethical issues in computer science. **The case study for SL is a scenario that includes two challenge questions that stimulate the required research.** The information obtained will prepare students to answer the questions in this section of the examination.

The case study will be published on the Programme Resource Centre 12 months before the May examination session (18 months before the November session). This will allow students to carry out detailed research prior to the examination and teachers to integrate the case study into their curriculum.

Through investigating the case study students can:

- **show** an understanding of how the system(s) in the case study work (AO1)
- **show** an understanding of the computational thinking fundamental to the system(s) in the case study (AO1)
- **apply** concepts from the course syllabus in the context of the case study (AO2)
- **explain** how scenarios in the case study may be related to other scenarios (AO2)
- **discuss** the impacts and ethical issues relevant to the case study (AO3)
- **evaluate, formulate and justify** strategies based on the information from the case study itself, their own research and new stimulus material provided in the examination paper (AO3).

A student's own research for the case study can include primary and secondary sources, interviews, guest speakers and field trips.

### Paper 2

**Duration: 1 hour 15 minutes**

**Weighting: 35%**

**Marks: 50**

Paper 2 consists of extended-response questions linked to SL topics in theme B: Computational thinking and problem-solving.

All questions on paper 2 are compulsory.

One of the questions will focus on algorithmic thinking without the need to interpret or write code.

There are two versions of paper 2—one for students who have studied Python, one for students who have studied Java.

The Python programming language has a number of built-in functions that can help with finding solutions to common computational challenges. The use of functions such as sort, pop, len, max and min can be very useful when programming algorithms and their use is encouraged in tasks such as the IA (the computational solution). However, in order to assess the full range of students' abilities, certain questions in the examination will prohibit specific built-in functions. These instances will be clearly indicated in the examination papers.

## External assessment details—HL

### Paper 1

**Duration: 2 hours**

**Weighting: 40%**

**Marks: 80**

Paper 1 consists of two sections. All questions on paper 1 are compulsory.

**Section A (56 marks)**

Section A consists of extended-response questions linked to the SL and HL topics on theme A: Concepts of computer science.

Questions will be common to SL with additional question parts that assess HL-only topics.

**Section B (24 marks)**

Section B consists of short- and extended-response questions examining concepts of computer science based on the pre-seen case study and the four challenge questions given in the case study.

| Section A and section B are completed together without any interruptions. |
|---|

#### Case study

The computer science case study provides the stimulus to investigate a scenario involving current developments, emerging technologies and/or ethical issues in computer science. **The case study for HL is a scenario that includes four challenges that stimulate the required research.** The information obtained will prepare students to answer the questions in this section of the examination.

The case study will be published on the Programme Resource Centre 12 months before the May examination session (18 months before the November session). This will allow students to carry out detailed research prior to the examination and teachers to integrate the case study into their curriculum.

Through investigating the case study students can:

- **show** an understanding of how the system(s) in the case study work (AO1)
- **show** an understanding of the computational thinking fundamental to the system(s) in the case study (AO1)
- **apply** concepts from the course syllabus in the context of the case study (AO2)
- **explain** how scenarios in the case study may be related to other scenarios (AO2)
- **discuss** the impacts and ethical issues relevant to the case study (AO3)

- **evaluate, formulate and justify** strategies based on the information from the case study itself, their own research and new stimulus material provided in the examination paper (AO3).

A student's own research for the case study can include primary and secondary sources, interviews, guest speakers and field trips.

## Paper 2

**Duration: 2 hours**

**Weighting: 40%**

**Marks: 80**

Paper 2 consists of extended-response questions linked to the SL and HL topics in theme B: Computational thinking and problem-solving.

All questions on paper 2 are compulsory.

Two questions from the SL paper will be repeated in the HL paper, including a question with a focus on algorithmic thinking without the need to interpret or write code.

Additional questions will focus on HL-only topics.

There are two versions of paper 2—one for students who have studied Python, one for students who have studied Java.

The Python programming language has a number of built-in functions that can help with finding solutions to common computational challenges. The use of functions such as sort, pop, len, max and min can be very useful when programming algorithms and their use is encouraged in tasks such as the IA (the computational solution). However, in order to assess the full range of students' abilities, certain questions in the examination will prohibit specific built-in functions. These instances will be clearly indicated in the examination papers.

# Internal assessment

## Purpose of internal assessment

IA is an integral part of the course and is compulsory for both SL and HL students. The assessment criteria are identical for both SL and HL students.

The IA task for computer science is a computational solution. This component brings together the two main themes of the syllabus in a single task where students can demonstrate their knowledge of, and ability to apply, the computational thinking process.

For the computational solution, students:

- select a topic within computer science
- specify a problem of their own choosing
- create a computational solution to this problem that demonstrates their skills and knowledge of the computational thinking process and of their chosen topic.

The IA task should, as far as possible, be woven into normal classroom learning and teaching and not be a separate activity that is unrelated to other elements of the course.

Students can choose their problem from a wide range of contexts, and it should be of personal interest to them.

During their work on the computational solution, students can demonstrate their knowledge of processes such as decomposition, pattern recognition, algorithmic thinking, writing programs, debugging and testing. This task allows them to do this without the time limitations and other constraints associated with written examinations.

The final product of the computational solution is a document that defines the problem and clearly demonstrates the solution using computational thinking in line with the assessment criteria. This will also be accompanied by a video where the student demonstrates the functionality of their solution and examples of their testing strategy.

## Guidance and authenticity

The computational solution submitted for assessment must be the student's own work. However, it is not the intention that students should decide upon a title or topic and be left to work on the IA component without any further support from the teacher. The teacher should play an important role during both the planning stage and the period when the student is working on the internally assessed work.

It is the responsibility of the teacher to ensure that students are familiar with the:

- requirements of the type of work to be internally assessed
- assessment criteria—students must understand that the work submitted for assessment must address these criteria effectively.

Teachers and students must discuss the IA work. Students should be encouraged to initiate discussions with the teacher to obtain advice and information, and must not be penalized for seeking guidance. As part of the learning process, teachers should read and give advice to students on one draft of the work. The teacher should provide oral or written advice on how the work could be improved, but not edit the draft. The next version handed to the teacher must be the final version for submission.

It is the responsibility of teachers to ensure that all students understand the basic meaning and significance of concepts that relate to academic integrity, especially authenticity and intellectual property. Teachers must ensure that all student work for assessment is prepared according to the requirements and must explain clearly to students that the IA work must be entirely their own. Where collaboration between students is permitted, it must be clear to all students what the difference is between collaboration and collusion.

All work submitted to the IB for moderation or assessment must be authenticated by a teacher, and must not include any known instances of suspected or confirmed malpractice. Each student must confirm that the work is their authentic work and constitutes the final version of that work. Once a student has officially submitted the final version of the work, it cannot be retracted. The requirement to confirm the authenticity of work applies to the work of all students, not just the sample work that will be submitted to the IB for the purpose of moderation.

For further details, refer to the IB publications *Academic integrity policy*, *Diploma Programme: From principles into practice* and the relevant general regulations in *Diploma Programme Assessment procedures* (updated annually).

Authenticity may be checked by discussion with the student on the content of the work, and scrutiny of the following.

- The student's initial proposal
- The first draft of the written work
- The references cited
- The style of writing compared with work known to be that of the student
- The analysis of the work by a web-based plagiarism detection service such as turnitin.com

The same piece of work cannot be submitted to meet the requirements of both the IA and the EE.

## Time allocation

IA is an integral part of the computer science course, contributing 30% to the final assessment in the SL course and 20% in the HL course.

It is recommended that 35 hours of teaching time should be allocated to the work. This should include:

- time for the teacher to explain to students the requirements of IA
- class time for students to work on the IA component and ask questions
- time for consultation between the teacher and each student
- time to review and monitor progress, and to check authenticity.

## Requirements and recommendations

Teachers and students will need to discuss issues relating to the problem the student wants to solve, strategies for developing a solution, and methods for testing and evaluating their solution. Students should be encouraged to use skills of inquiry and research to initiate discussions with the teacher to obtain advice and information, and will not be penalized for seeking support.

## Ethical guidelines

Given the nature of the IA task, students must take into account ethical problems and implications concerning undertaking research and developing the solution. For example, they should ensure the confidentiality and security of data. Wherever possible, original data should be used or collected by the student.

The following guidelines must be applied.

- Consent must be obtained from people who will be involved in the development of the computational solution before any investigation begins.
- Written consent must be obtained from the owner of any existing system that will be used as part of the IA—for example, when implementing a security analysis protocol on an existing system.
- All data collected must be stored securely to maintain confidentiality.

- Data collected can only be used for the computational solution. It must not be used for any other purpose without explicit permission.

## Health and safety guidelines

Schools are advised to follow local best practice in health and safety for research linked to the development of the computational solution. Each school is ultimately responsible for the health and safety of students.

## Using assessment criteria for internal assessment

For IA, a number of assessment criteria have been identified. Each assessment criterion has level descriptors describing specific achievement levels, together with an appropriate range of marks. The level descriptors concentrate on positive achievement, although for the lower levels failure to achieve may be included in the description.

Teachers must judge the internally assessed work at SL and at HL against the criteria using the level descriptors.

- The same assessment criteria are provided for SL and HL.
- The aim is to find, for each criterion, the descriptor that conveys most accurately the level attained by the student, using the best-fit model. A best-fit approach means that compensation should be made when a piece of work matches different aspects of a criterion at different levels. The mark awarded should be one that most fairly reflects the balance of achievement against the criterion. It is not necessary for every single aspect of a level descriptor to be met for that mark to be awarded.
- When assessing a student's work, teachers should read the level descriptors for each criterion until they reach a descriptor that most appropriately describes the level of the work being assessed. If a piece of work seems to fall between two descriptors, both descriptors should be read again and the one that more appropriately describes the student's work should be chosen.
- Where there are two or more marks available within a level, teachers should award the upper marks if the student's work demonstrates the qualities described to a great extent; the work may be close to achieving marks in the level above. Teachers should award the lower marks if the student's work demonstrates the qualities described to a lesser extent; the work may be close to achieving marks in the level below.
- Only whole numbers should be recorded; partial marks (fractions and decimals) are not acceptable.
- Teachers should not think in terms of a pass or fail boundary but should concentrate on identifying the appropriate descriptor for each assessment criterion.
- The highest level descriptors do not imply faultless performance but should be achievable by a student. Teachers should not hesitate to use the extremes if they are appropriate descriptions of the work being assessed.
- A student who attains a high achievement level in relation to one criterion will not necessarily attain high achievement levels in relation to the other criteria. Similarly, a student who attains a low achievement level for one criterion will not necessarily attain low achievement levels for the other criteria. Teachers should not assume that the overall assessment of the students will produce any particular distribution of marks.
- It is recommended that the assessment criteria be made available to students.

## Internal assessment details—SL and HL

**Duration: 35 hours**

**Weighting: SL 30%, HL 20%**

## Introduction

The IA task requires the student to identify a problem of their own choosing and develop a software solution for it, using the computational thinking process.

The solution is assessed using five criteria.

- Criterion A: Problem specification
- Criterion B: Planning
- Criterion C: System overview
- Criterion D: Development
- Criterion E: Evaluation

## Key terms

**Solution:** This refers to the documentation and video submitted by the student for the IA.

**Product:** This refers to the completed software only.

## Choice of problem

In identifying a problem, the student can select to apply to the problem any topic in computer science that interests them. It does not have to be directly related to the specified themes in the syllabus.

The problem chosen should require a software solution of sufficient complexity to be commensurate with the level of this DP computer science course. It should also require sufficient innovation for the student to demonstrate their organizational skills, algorithmic thinking and ability to code their algorithms.

More examples and detail of the features of complex and innovative solutions are given in the *Computer science teacher support material*.

## The nature of the solution

All solutions must be coded and can take a number of forms. These can include:

- creating a new system, such as an OOP program, interactive web-based application using a database, computer game, mobile application, simulation, stand-alone application, web-based application
- adding functionality to an existing system, such as connecting a webpage(s) to a database, writing a function for Moodle, writing a plug-in, or developing a stand-alone application.

Whichever problem and form of solution the student chooses, it is essential that they explicitly demonstrate and document their algorithmic thinking skills. Products that take existing templates that show no evidence of modification in their structure, design or functionality are not appropriate. Examples of **inappropriate** products include:

- the development of a programming product using only copied code
- the development of a website (product) using a web-based template that predetermines its structure and layout
- the use of exemplar products or templates provided with software, (e.g. the Northwind database in Microsoft Access)
- a copied computer game without major modifications to the code that have been properly documented
- a product that does not meet the ethical requirements outlined in the "Requirements and recommendations" section of this publication
- a computer/mobile application created using a builder/wizard/drag-and-drop tool without the need for code development.

## Requirements

The IA submission consists of three types of files.

- Documentation

- Video
- Appendices

### Documentation

The documentation must be submitted as a single PDF file.

The documentation must include five separate sections, one for each of the five criteria.

The total word count for the documentation must not exceed **2,000 words**. This does not include excerpts of code, comments or diagrams. The overall word count must be clearly written on the first page of the document.

More detail on the nature of the code excerpts, and related comments, are given in the *Computer science teacher support material*.

### Video

The purpose of the video is to provide evidence of the functionality of the product and to give examples of the testing strategy. The video must be submitted as a separate file. It must also:

- be no longer than five minutes
- be in a commonly used format such as mp4, .avi or .wmv
- demonstrate the full functionality of the product
- demonstrate examples of the testing strategy used in the development of the product.

More support on the nature and making of the video can be found in the *Computer science teacher support material*.

### Appendices

Appendices must be submitted as a single PDF file. The appendices must include the full source code and any other resources developed by the student that are referred to in the documentation.

The appendices are **not** used as evidence for awarding marks, and examiners are not required to read the appendices. However, solutions that do not include an appendix with the full source code cannot be awarded full marks for the techniques demonstrated in criterion D.

# Task descriptors and criteria—SL and HL

The computer science IA focuses on using the computational thinking processes and the skills of algorithmic thinking and programming to solve a problem chosen by the student. The computational solution is about problem-solving.

## The assessment criteria

Criteria A, B, C and E are process-oriented. The criteria both guide and assess how the IA task is carried out and allow common assessment criteria to be applied to different types of products.

Criterion D is an assessment of the final product and assesses the student’s understanding of the concepts involved in its development. The ability to carry out a testing strategy on the final product is a key element in this criterion. Examples from the testing strategy should be demonstrated in the video and will be used as part of the evidence for this criterion.

### Criterion A: Problem specification (4 marks)

The problem specification is the starting point of the solution and must be used as a basis for the development of the product.

- The student should have the necessary technical skills, access to appropriate hardware and software, and availability of relevant data to address the problem.
- The success criteria identified in the problem specification (criterion A) will be used in the planning (criterion B), in the development (criterion D) and in the evaluation (criterion E).

The recommended word count for this criterion is **300 words**.

| Marks | Level descriptor |
|---|---|
| 0 | The response does not reach a standard described by the descriptors below. |
| 1–2 | The response:<br>• **outlines** a problem scenario<br>• **states** limited success criteria<br>• **outlines** the nature of the solution in a computational context. |
| 3–4 | The response:<br>• **describes** the problem scenario in terms of its measurable solution requirements<br>• **states** appropriate success criteria<br>• **explains** the choice of computational context for the solution. |

| Problem specification clarifications |
|---|
| The **problem scenario** is a clear description of the problem including its measurable solution requirements. The description should relate directly to the problem, whether this be in the world around us, other fields of knowledge, or a current issue in computing.<br>**Success criteria** are measurable outcomes derived from the solution requirements that indicate the successful development of the product.<br>The **computational context** is the specific area of computing that is selected to be used in the solution. |

### Criterion B: Planning (4 marks)

The planning of the product must be consistent with the problem specification in criterion A.

- This criterion assesses how the problem scenario has been decomposed into component parts.
- The plan should address the requirements of the solution in terms of the success criteria, and include a proposed chronology for the steps involved in planning, designing, developing, testing and evaluating the solution.
- A plan can be presented in different forms, but popular diagram formats such as GANTT and AGILE charts can effectively support the planning process.
- The plan may include any relevant research, such as the use of existing code libraries.

The recommended word count for this criterion is **150 words**.

| Marks | Level descriptor |
|---|---|
| 0 | The response does not reach a standard described by the descriptors below. |
| 1–2 | The response:<br>• **constructs** a partial decomposition of the problem scenario<br>• **constructs** a plan that addresses some of the success criteria of the solution. |
| 3–4 | The response:<br>• **constructs** a reasonable decomposition of the problem scenario<br>• **constructs** a plan that addresses the success criteria of the solution. |

| Planning clarifications |
|---|
| **Decomposition** is the breaking down of the problem scenario identified in criterion A into smaller, more manageable sub-problems or components. The decomposition can be effectively constructed using diagrams.<br>A **reasonable decomposition** breaks the problem down into essential components that support the construction of a plan. |

## Criterion C: System overview (6 marks)

The system overview of the product must be consistent with the problem specification in criterion A and the planning in criterion B.

- The system overview should include a system model with the key components, their relationships, the rules governing their interaction, and the algorithms required by these components and the user interface.
- The system overview should have the clarity to enable a third party to recreate the product.
- The system model will provide the information for a viable testing strategy.

The recommended word count for this criterion is **150 words**.

| Marks | Level descriptor |
|---|---|
| 0 | The response does not reach a standard described by the descriptors below. |
| 1–2 | The response:<br>• **outlines** a limited system model<br>• **identifies** algorithms for the components of the system model<br>• **identifies** a testing strategy for at least one success criterion. |
| 3–4 | The response:<br>• **constructs** a system model that is not complete<br>• **constructs** algorithms for the components of the system model that lead to partial functionality of the product<br>• **outlines** a testing strategy that aligns with at least three success criteria. |
| 5–6 | The response:<br>• **constructs** a complete system model<br>• **constructs** algorithms for the components of the system model that enable the product to perform<br>• **describes** a testing strategy that aligns with the success criteria. |

| System overview clarifications |
|---|
| A **system model** consists of diagrams that include the components of the system and how they are connected. The system model will include the design of the user interface. A **complete system model** does not include the algorithms for each of the components.<br>**Algorithms** can be presented in different forms, including natural language, flow charts or pseudocode, and should address the individual components of the system model.<br>The **testing strategy** refers to a systematic approach for evaluating whether the computational solution works as intended. The testing strategy should ensure that code functions correctly and handles |

| System overview clarifications |
|---|
| unexpected or incorrect inputs. This can be represented effectively in a table with proposed test data and expected outcomes. |

## Criterion D: Development (12 marks)

The development of the product must be consistent with the problem specification in criterion A, the planning in criterion B and the system overview developed in criterion C.

- The video must provide evidence of the functionality and give examples of the testing of the product.
- The development of the solution must justify the structure of the product, why the structure is appropriate, and demonstrate the techniques used to develop the product based on the algorithms constructed in criterion C. These techniques may include loops, data structures, existing libraries and the integration of software tools.
- The testing strategy must include testing for correctness, reliability and efficiency. The testing must be described and justified in the documentation with supporting examples seen in the video.

The recommended word count for this criterion is **1,000 words**.

| Marks | Level descriptor |
|---|---|
| 0 | The response does not reach a standard described by the descriptors below. |
| 1–3 | The response:<br>• **constructs** a product with very limited functionality<br>• **constructs** a product using no appropriate techniques to implement the algorithms<br>• **states** the choices made to implement the algorithms<br>• **states** the testing strategy used. |
| 4–6 | The response:<br>• **constructs** a product that has limited functionality<br>• **constructs** a product using at least one appropriate technique to implement the algorithms<br>• **outlines** the choices made to implement the algorithms<br>• **states** the effectiveness of the testing strategy. |
| 7–9 | The response:<br>• **constructs** a product that has partial functionality<br>• **constructs** a product that uses some appropriate techniques to implement the algorithms<br>• **explains** the choices made to implement the algorithms<br>• **describes** the effectiveness of the testing strategy. |
| 10–12 | The response:<br>• **constructs** a fully functional product<br>• **constructs** a product that uses appropriate techniques to implement the algorithms<br>• **evaluates** the choices made to implement the algorithms<br>• **justifies** the effectiveness of the testing strategy. |

| Development clarifications |
|---|
| **Implementation and coding of the algorithms:** Techniques in the criteria refer to the process of programming algorithms using code. The documentation must highlight key elements of code that are important for the efficient functioning of the algorithms. Any code presented in the solution must include relevant comments, be consistent and be readable. Code excerpts included in the documentation must be referenced to the full source code submitted as an appendix.<br>The video must demonstrate the **functionality** of the product. The deployment of the **testing** strategy and its effectiveness must be described in the documentation, with examples of the testing seen in the video. |

## Criterion E: Evaluation (4 marks)

The evaluation of the product must be consistent with the problem specification and success criteria in criterion A.

The recommended word count for this criterion is **400 words**.

| Marks | Level descriptor |
|---|---|
| 0 | The response does not reach a standard described by the descriptors below. |
| 1–2 | The response:<br>• **states** the extent to which the success criteria were met<br>• **describes** improvements to the product. |
| 3–4 | The response:<br>• **evaluates** the extent to which the success criteria were met<br>• **justifies** improvements to the product. |

# Command terms

Students should be familiar with the following key terms and phrases used in examination questions, which are to be understood as described below. Although these terms will be used frequently in examination questions, other terms may be used to direct students to present an argument in a specific way.

| Term | Objective | Definition |
|---|---|---|
| **Calculate** | 2 | Obtain a numerical answer showing the relevant stages in the working. |
| **Compare** | 3 | Give an account of the similarities between two (or more) items or situations, referring to both (all) of them throughout. |
| **Construct** | 3 | Display information in a diagrammatic or logical form. |
| **Deduce** | 3 | Reach a conclusion from the information given. |
| **Define** | 1 | Give the precise meaning of a word, phrase, concept or physical quantity. |
| **Describe** | 2 | Give a detailed account. |
| **Discuss** | 3 | Offer a considered and balanced review that includes a range of arguments, factors or hypotheses. Opinions or conclusions should be presented clearly and supported by appropriate evidence. |
| **Distinguish** | 2 | Make clear the differences between two or more concepts or items. |
| **Estimate** | 2 | Obtain an approximate value. |
| **Evaluate** | 3 | Make an appraisal by weighing up the strengths and limitations. |
| **Explain** | 3 | Give a detailed account including reasons or causes. |
| **Identify** | 2 | Provide an answer from a number of possibilities. |
| **Justify** | 3 | Give valid reasons or evidence to support an answer or conclusion. |
| **Label** | 1 | Add labels to a diagram. |
| **List** | 1 | Give a sequence of brief answers with no explanation. |
| **Outline** | 2 | Give a brief account or summary. |
| **Sketch** | 3 | Represent by means of a diagram or graph (labelled as appropriate). The sketch should give a general idea of the required shape or relationship, and should include relevant features. |
| **State** | 1 | Give a specific name, value or other brief answer without explanation or calculation. |
| **Suggest** | 3 | Propose a solution, hypothesis or other possible answer. |
| **To what extent** | 3 | Consider the merits or otherwise of an argument or concept. Opinions and conclusions should be presented clearly and supported with appropriate evidence and sound argument. |
| **Trace** | 2 | Follow and record the actions of an algorithm. |

# Bibliography

Dijkstra, E. W. (1976). *A programmer's early memories* [Video]. YouTube. https://youtu.be/L5EyOokcl7s?

Wing, J. M. (2014, January 10). Computational thinking benefits society. *Social Issues in Computing*. University of Toronto. http://socialissues.cs.toronto.edu/index.html%3Fp=279.html