Diploma Programme

Computer science teacher support material

International Baccalaureate®
Baccalauréat International
Bachillerato Internacional

Diploma Programme

Computer science teacher support material

International Baccalaureate®
Baccalauréat International
Bachillerato Internacional

Diploma Programme
Computer science teacher support material

Published March 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.

International Baccalaureate®
Baccalauréat International
Bachillerato Internacional

The purpose of this teacher support material

Welcome to the Diploma Programme (DP) computer science teacher support material (TSM). This TSM is designed to assist both new and experienced teachers in building or revising their course design. It is intended to add insight, inspiration and guidance to the teacher and student journey by:

- supporting experienced and inexperienced teachers in structuring and delivering the course
- supporting teachers with developing students' abilities in computational thinking
- supporting teachers with developing students' practical computing skills in algorithmic thinking and programming
- complementing IB professional development (PD).

The TSM is structured to cover key programme pedagogy such as the approaches to learning and approaches to teaching and how these relate to computer science, as well as subject-specific considerations such as developing computational thinking and the practical skills of computer science.

Acknowledgement

The International Baccalaureate (IB) would like to thank the educators who have contributed time and resources to the development of the DP *Computer science guide* and TSM.

Course overview

The DP computer science course is organized into two themes: theme A: Concepts of computer science, and theme B: Computational thinking and problem-solving. While the two themes are different, they both support an understanding of the computational thinking process. The TSM aims to demonstrate how students can learn about this process through the two themes. The TSM also supports the development of students' practical skills of algorithmic thinking and programming through the course content.

Theme A covers the development of concepts in computing. It explores how computers operate, from simple binary computations to computing systems that allow machine learning to take place. Theme B covers the development of techniques in computing that facilitate problem-solving through the automation of algorithms with computer programs.

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

Theme A: Concepts of computer science

The course begins by looking inside the "black box" of computers, unravelling their inner workings. Students embark on a journey into the intricate realm of logic gates, circuitry, central processing units

(CPUs) and signals, bridging the divide between abstract computational thinking and the practical development of everyday programs.

Having developed a solid understanding of the fundamentals, students extend their knowledge beyond the confines of the computer, exploring how network technology empowers communication, research, collaboration, organization and the sharing of ideas. They gain insights into the many facets of networks, from the transmission of pulses of light through fibre optic cables, to the protocols enabling the transfer of vast data volumes over the internet via undersea cables connecting continents.

Our daily activities generate substantial amounts of data, a precious resource that—when properly stored and made accessible—offers solutions and insights to address a multitude of challenges. Therefore, students delve into the storage of data on databases, its widespread accessibility, security measures and how this technology can simultaneously enable ubiquitous global access to information while protecting private information from unauthorized use.

Enormous aggregations of data, often referred to as "big data", undergo meticulous analysis to unveil patterns and predict future behaviours. Therefore, it is important for students to study how machine learning harnesses this data to construct models for diverse applications, from self-driving cars to artificial intelligence (AI)—a transformative development in computing that will increasingly shape their lives. Both standard level (SL) and higher level (HL) students learn about the intricacies of machine learning, its various forms and, importantly, the ethical considerations surrounding its application in their futures.

The development of an understanding of the concepts of computer science should go hand in hand with the development of algorithmic thinking and programming skills. This knowledge empowers students to tackle problems and explore specific course areas through computational thinking, guided by their individual interests and passions. The internal assessment (IA) component allows students to show what they have learned through their computational solution.

Theme B: Computational thinking and problem-solving

Computational thinking is the organizing concept of the DP computer science course and is defined in detail in the *Computer science guide*. The overarching goal is to solve real-world problems from different subject areas or within computing. In this course, the computational thinking process is the conceptual approach by which these problems are solved.

There are different definitions of computational thinking that, while sharing the same key features, emphasize specific components or use different language for the process. It is important that teachers carefully read the section on computational thinking in the *Computer science guide* to be clear about the specific definition of the process used in this course.

The IA computational solution requires students to solve a problem that they have chosen for themselves using the computational thinking process. The IA criteria reflect the steps in the process, so it is vital that students develop the ability to think computationally to complete the computational solution task successfully and efficiently. There is a short topic dedicated to computational thinking in theme B, however, it is strongly recommended that computational thinking is the lens through which content is delivered, and that knowledge and understanding of the process of computational thinking is acquired by students throughout the course.

As such, this TSM provides examples and support for teachers to develop computational thinking in their classroom through the delivery of course content in theme A and theme B.

Practical skills in computer science

The *Computer science guide* classifies the skills of computer science as both technical skills and personal skills. In the guide, the personal skills are covered in detail in "The approaches to learning framework". The technical skills of algorithmic thinking and programming are introduced as follows and then developed in detail in examples of how they can be developed through the course content. Students may come to this course with little or no knowledge of programming, while others may have many years of experience. Suggested approaches to develop students' technical skills are given in specific sections of this TSM.

Algorithmic thinking

Algorithmic thinking is the focused practice of creating, analysing and implementing algorithms to solve problems with efficiency and effectiveness. This course focuses on enabling students to design algorithms, trace their execution and evaluate their performance; thus honing skills that extend into logical structuring and the development of systematic solutions. The emphasis is on understanding algorithms' operational logic and applying this understanding across various contexts, using tools such as pseudocode and flow charts for representation. Mathematical reasoning supports this process by providing a framework for structured problem-solving, but it is not the primary focus. Throughout the TSM and practical course activities, students are encouraged to apply algorithmic principles in real-world scenarios, ensuring a comprehensive grasp of both the theoretical foundations and practical applications of algorithmic thinking.

Programming

A fundamental skill in computer science is the ability to program a computer. This course aims to cover the process of transforming algorithms into computer code in the context of the computational thinking process and solving problems. Programming is an element in a process—it is not an end in itself. Similarly, computer code is simply a means to automate an algorithmic process in a machine. This automation allows computations to be carried out at a level of complexity and efficiency that is not possible for a human.

The course can be studied using either Java or Python. There is no advantage or disadvantage in using one or other of these programming languages. The choice will be determined by the prior experience of students and the preference of teachers.

The skills assessed in the course are not dependent on a specific language and assessment reflects this. Paper 2 has two versions: one for students who studied in Java, the other for students who studied in Python. Both versions of paper 2 assess the same skills in the same contexts.

It is acknowledged that Python has a number of useful in-built functions, and these could be used in IA. However, in external examination it will be made clear when an in-built function from Python should not be used.

Teaching sequence

When structuring learning and teaching of a course, especially one as comprehensive as computer science, the scope and sequence in which units are presented can significantly impact student understanding, engagement and overall learning outcomes. While it is entirely feasible to deliver the course content in a linear fashion—starting with theme A: Concepts of computer science and concluding with theme B: Computational thinking and problem-solving—such an approach may not leverage the full potential and synergy of the topics, nor maintain a high level of student engagement. A more effective strategy involves integrating the practical and theoretical aspects throughout the course, blending themes A and B from the outset. This method promotes a deeper understanding by allowing students to see the immediate application of theoretical concepts in practical scenarios.

It is suggested to combine the theoretical knowledge found in theme A with the practical exercises of theme B at strategic points. This approach will ensure that as complexity increases, students remain engaged, able to see the relevance of what they are learning to real-world applications, and naturally make connections between theory and practice. This fosters a learning environment where theoretical insights enhance practical skills, and vice versa, facilitating a more holistic grasp of computer science.

A possible order of units that aims to optimize learning by integrating the conceptual and application-based elements of the curriculum is given in the section "Developing a scope and sequence". This proposed sequence is designed to build upon foundational knowledge gradually, introduce students to increasingly complex concepts in a manageable and engaging way, and ensure that practical programming skills are developed in tandem with theoretical understanding.

Note that the computer science case study assessed in paper 1 changes every year and is released 12 months before the May examination session. The November session will use the same case study as the May session of that calendar year. The case study provides a scenario in which to frame several of the theory topics and can be introduced gradually throughout theme A, building understanding until it is formally covered in full detail.

Syllabus structure and features

Guiding questions

Each topic starts with a guiding question. Students may be able to answer the questions in different ways at different stages of their learning, with increasing depth and breadth as their understanding of the topic develops.

The guiding questions can be used to support learning and teaching as:

- **openers** for a topic or subtopic
- suggestions for an **overview** of content and **assessment** of learning
- **stimuli** to generate further guiding questions.

Each of these approaches is explored as follows, using different examples of guiding questions from the guide.

Guiding questions as openers for a topic or subtopic

An example is the following guiding question from "A4 Machine learning".

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

The question can be used in several ways, especially for subtopic "A4.4.1 Discuss the ethical implications of machine learning in real-world scenarios".

Prior knowledge (initiating the subtopic)

Introduction: Begin with a review of fundamental machine learning concepts and the recognition of patterns in data. Discuss the importance of data integrity, algorithm transparency and the potential for bias in model training. This sets the stage for understanding the ethical implications of machine learning applications.

Ideas for opening discussions (during the subtopic introduction)

Opening question: How might the data used to train machine learning models influence their fairness and accuracy?

Case studies: Present real-world scenarios where machine learning has led to ethical dilemmas, such as biased hiring practices or surveillance overreach. These discussions can help students appreciate the complexity of ethical considerations in technology.

Developing understandings (during the learning of the subtopic)

Group activities: Divide students into groups to explore different ethical issues related to machine learning, such as privacy concerns, data manipulation and algorithmic bias. Each group could focus on a specific area, research real-life incidents and propose solutions or guidelines.

Expert opinions: Incorporate insights from ethicists, data scientists and technologists on best practices for ethical machine learning. This could include guest lectures or curated video content.

Extending learning (at the end of the subtopic)

Project-based learning: Encourage students to develop their machine learning models on data sets, and then critically analyse them for potential biases and ethical implications. This hands-on approach can help solidify the theoretical knowledge gained during the course.

Ethical framework proposal: As a culmination of their learning, students could work in teams to draft an ethical framework for developing and deploying machine learning models. This framework should consider accuracy, fairness, transparency and accountability.

Guiding questions as suggestions for an overview of content and assessment of learning

Using the same example from "A4 Machine learning", students can use various methods—such as discussion and debate, presentations, essay writing—to answer the guiding question about ensuring ethical accuracy in machine learning models. This will involve reflecting on their learning about ethical considerations and the impact of biases.

Reviewing the topic encourages students to critically evaluate the use of data, algorithmic fairness and the societal implications of machine learning, reinforcing the importance of ethical practices. By ensuring students focus on specific elements such as data integrity and algorithm transparency, teachers can guide students to deepen their understanding of how to approach machine learning ethically.

While answering this guiding question, additional examples and emerging issues in the field can be introduced, broadening perspective on the application of ethical principles in diverse machine learning scenarios. Such examples and issues might include algorithmic transparency in criminal justice, ethical AI in employment, fairness in financial services, privacy concerns in surveillance technologies, ethical machine learning in healthcare, bias in AI-driven content recommendation systems, sustainable AI and environmental impact, and the role of AI in global inequalities.

Guiding questions as stimuli to generate further guiding questions

The guiding question from "A4 Machine learning" can be used to generate further guiding questions, and hence generate more interest and prompt further research in the topic. An approach to this process might be to begin by examining specific instances where ethical dilemmas arise in AI implementation. Each case study or example could be analysed to understand the underlying ethical principles at play and their implications for society. This encourages critical thinking by posing probing questions and fostering discussions around the ethical implications of AI technologies.

For example, by looking at the issues and examples generated in the previous section, the following questions might be generated by students and are possible further guiding questions.

- How can algorithmic transparency be ensured in AI-driven decision-making processes, particularly in sensitive areas such as criminal justice?
- What ethical guidelines should govern the use of AI in employment practices to ensure fairness and prevent discrimination?
- How can financial institutions mitigate biases in AI algorithms to promote equitable access to financial services?
- What measures can be implemented to safeguard individual privacy in the face of surveillance technologies powered by AI?
- In what ways can machine learning algorithms be ethically designed and deployed to enhance healthcare outcomes while respecting patient autonomy?
- How can content recommendation systems be optimized to minimize algorithmic bias and promote diverse perspectives?
- What strategies can be adopted to develop environmentally sustainable AI solutions and mitigate their potential negative impact on the environment?
- What role does AI play in perpetuating or alleviating global inequalities, and how can ethical AI frameworks address these disparities?

Linking questions

Every topic has a set of linking questions that connect to other topics in the syllabus, to theory of knowledge (TOK), to other courses in the DP and to real-world contexts. In seeking to answer these questions, students are encouraged to make connections across the syllabus to illustrate the interconnectedness of concepts.

Linking questions as connections between different topics

In the DP computer science course, the topics and concepts are designed to provide a holistic view of the field. Each segment—from core computer science concepts to advanced topics and applied areas—is not only foundational but also interlinked. These connections facilitate a deeper understanding and practical application of computer science, enabling students to see the relevance and impact of each topic in the broader context of the field. As students explore these diverse areas, ranging from hardware and operation to cybersecurity and cloud computing, the curriculum reveals how each part complements and enhances the others, forming an integrated and comprehensive educational experience.

Questions are helpful in establishing these links and can help students build conceptual understanding across the course. Students are also encouraged to ask their own linking questions where possible. The guide includes examples of these questions, and some additional examples are presented as follows.

- How does understanding computer hardware and operation (A1.1) contribute to the development of efficient algorithms and programming solutions (B2.4)?
- What role do network fundamentals (A2.1) play in ensuring the security of data transmissions (A2.3)?
- How do the fundamentals of programming (B2.1) influence the design and implementation of databases (A3.2)?
- How do concepts of object-oriented programming (OOP) (B3) enhance the development of machine learning algorithms (A4.1)?
- What ethical considerations (A4.4) arise in the context of machine learning approaches (A4.3)?

Linking questions for TOK

Computational thinking enhances TOK by offering a methodology for understanding and linking various disciplines. It applies techniques like pattern recognition to diverse fields, from algorithmic applications in computer science to data analysis in history. In areas like bioinformatics, the role of computational thinking in analysing DNA for disease research showcases its practical impact. Similarly, its use in data science, AI and digital humanities highlights the importance of computational thinking in generating new knowledge and encouraging critical reflection across disciplines.

In computer science, links to TOK can be made through questions. The TOK course includes a theme of technology, and this can be a helpful stimulus to produce linking questions to TOK. The following are some examples of questions linking computer science to TOK.

- Can computational thinking be considered the universal language of knowledge?
- How do the fundamentals of computer hardware and operation relate to broader philosophical questions about the nature of humanity?
- How do network fundamentals contribute to our understanding of interconnectedness and communication within global systems?
- In what ways does the process of abstraction in programming mirror the human cognitive process of categorization and conceptualization?

Linking questions for other courses in the DP

In the DP, computer science stands out as a key interdisciplinary field, linking concepts from other courses. Its role goes beyond mere technical application—it fosters a deeper understanding and innovative approaches across diverse subjects. The following are some examples of questions linking computer science to other DP subjects.

- How do principles of aesthetics and artistic value in the visual arts relate to the outputs of generative AI?
- How do mathematical concepts taught in the DP curriculum underpin the cryptographic techniques used in computer science?
- How do the scientific method and models in natural sciences relate to the development of computational models in computer science?
- How does the use of machine learning for automated buying and selling of stocks and shares impact market stability and investor protection?

Linking questions to apply computer science in real-world contexts

The goal of computer science is to solve real-world problems and deal with real-world tasks. Student engagement can therefore be enhanced by asking questions that relate computer science to the world around them. The relevance of the course can be made evident to students by asking thoughtful questions that relate what they are learning in class to their lived experience.

The following are some examples of questions linking computer science to real-world contexts. The questions may seem quite abstract at first, but it is important to relate each question to real-world contexts by considering the link carefully, considering real-world applications and/or impacts, and exploring examples.

- How does understanding computer hardware and operation (A1.1) contribute to the development of efficient algorithms and programming solutions (B2.4)?

Link	The performance of algorithms is linked to the capabilities of the hardware on which they run. A solid grasp of hardware specifics enables programmers to optimize algorithms, considering factors such as processing power and memory constraints.
Real–world application	In fields like high-frequency trading, algorithmic efficiency is important for success. Understanding hardware limitations and capabilities can lead to significant gains in transaction speed and system responsiveness.
Example	Understanding the intricacies of CPU architecture, such as cache sizes and pipeline processing, can lead programmers to design algorithms that maximize data processing speeds and minimize latency. In high-frequency trading, where milliseconds can make a significant difference, algorithm efficiency is critical. Efficient algorithms can process large volumes of transactions quickly, leading to better market positioning and potential financial gains.

- What role do network fundamentals (A2.1) play in ensuring the security of data transmissions (A2.3)?

Link	Secure data transmission is heavily dependent on the robustness of the underlying network infrastructure. Knowledge of network fundamentals is crucial for identifying vulnerabilities and implementing secure transmission protocols.
Real–world application	In the context of internet banking, secure data transmission is vital. The application of network fundamentals ensures that financial transactions are both swift and secure, safeguarding against potential cyber threats.
Example	Mastery of network fundamentals is key to securing data transmissions. This involves a comprehensive understanding of network protocols, encryption techniques and firewall configurations. In internet banking, where secure transmission of sensitive financial data is paramount, applying strong encryption protocols like transport layer security (TLS) ensures data integrity and confidentiality. Recognizing and mitigating

	risks such as man-in-the-middle attacks or Structured Query Language (SQL) injection is also critical in maintaining the security of data as it traverses the network.

- How do the fundamentals of programming (B2.1) influence the design and implementation of databases (A3.2)?

Link	Programming principles lay the groundwork for database design and management. Understanding data structures and algorithms guides the efficient retrieval, storage and manipulation of data in databases.
Real–world application	The management of patient records in healthcare systems relies on well-designed databases. Efficient programming leads to faster access to critical patient information, thereby improving healthcare delivery.
Example	Knowledge of data structures, such as trees and hash tables, is vital in creating efficient indexing systems for databases, which in turn facilitate quicker data retrieval. In the healthcare industry, efficient database design ensures rapid access to patient records, improving response times in critical care scenarios. Understanding transactional control and concurrency in programming can help to maintain database integrity in multi-user environments.

- How do concepts of OOP (B3) enhance the development of machine learning algorithms (A4.1)?

Link	OOP offers a framework that is particularly well suited to building complex, modular and scalable machine learning models.
Real–world application	In technology companies, the application of OOP principles in developing machine learning models facilitates better code reusability and easier maintenance, enhancing the agility and adaptability of AI-driven solutions.
Example	OOP concepts like encapsulation allow for creating machine learning models that encapsulate specific functionalities, making the codebase more organized and maintainable. In developing a machine learning model for image recognition, OOP facilitates the creation of separate modules for image preprocessing, feature extraction and classification, each of which can be developed, tested and improved independently.

- What ethical considerations (A4.4) arise in the context of machine learning approaches (A4.3)?

Link	As machine learning models become more common, ethical implications of their deployment, such as bias, transparency and accountability, become increasingly important.
Real–world impact	In areas like facial recognition technology, ethical considerations are paramount to ensure these systems are fair and do not infringe individual rights or privacy.
Example	The deployment of machine learning models involves the challenge of algorithmic bias (i.e. ensuring that AI systems do not perpetuate existing societal biases). In machine learning applications such as recruitment, lending and law enforcement, biased algorithms can have significant negative impacts on individuals and communities.

	Transparency and accountability are also important. This includes the need for mechanisms that trace decisions made by AI systems back to their algorithms and training data, to ensure fairness and rectify any issues.

Computational thinking

Engaging with computational thinking

The organizing concept of the DP computer science course is computational thinking. Students need to engage with this concept from the beginning and throughout the course, and as part of their preparation for internal and external assessment. Teachers should read with care the relevant sections on computational thinking in the guide and reflect on how they have used this form of thinking in their own experience of computer science.

Students with little experience of computer science may be unfamiliar with computational thinking. The concept should be made a key part of as much of the course content as possible and referenced explicitly in the classroom.

Students who come to the course with more developed skills in computer science may already be aware of computational thinking. However, despite the fact that these students have developed skills in computing, they may not explicitly think computationally but instead take an intuitive approach to the subject. It is important that these more experienced students' skills are framed in the context of computational thinking and that they can explicitly recognize the processes they employ.

Whether they are new to computer science or already have some experience of programming, throughout the course students should develop the skills associated with the computational thinking process. This approach to problem-solving is the key to course success.

Examples of integrating computational thinking

There are several areas in which computational thinking integrates the curriculum. This integration encompasses both theoretical aspects in theme A and practical applications in theme B. For example, in the domain of "A1 Computer fundamentals", abstraction plays a crucial role, enabling students to grasp how a computer operates by understanding the individual components and their interconnections. This is complemented by the use of decomposition, which simplifies the computer into more manageable, understandable parts. Similarly, in the study of databases, decomposition is instrumental in constructing the underlying relational structure, distilling complex arrays of data and entities into a coherent schema.

In the practical parts of the course in theme B, the programming segment leverages both decomposition and algorithmic thinking. These methods allow students to dissect a problem, isolate its components and devise a step-by-step solution. Additionally, in the realm of OOP, abstraction and pattern recognition are vital. They facilitate the creation of classes and an object model that accurately reflects the domain being addressed, aiding in the development of a robust and effective software design.

The paper 1 case study also benefits from the principles of computational thinking. The specifics of the case study vary each year. However, the approaches to research and analysis consistently employ computational thinking techniques. For example, students might use decomposition to segment the study area into more manageable pieces, allowing different groups to concentrate on specific segments. In doing so, they can abstract their work from the broader context and focus intently on their assigned segment, while maintaining a clear understanding of how it fits into the overall case study. This multifaceted approach ensures that computational thinking is not only a foundational element of the curriculum but also a versatile tool that students can apply across a wide range of contexts and challenges.

Computational thinking and external assessment

Computational thinking is a fundamental skill set, especially relevant to paper 1 and paper 2. In paper 1, a question on the fetch–execute cycle requires students to decompose the cycle into understandable parts.

Similarly, when discussing how data is communicated across networks, abstraction allows students to focus on the essential elements of network protocols and data transmission processes, providing a clearer understanding of complex network operations. Questions on database design might call for an explanation of how databases maintain atomicity, consistency, isolation, durability (ACID) properties in transactions. This clearly involves both algorithmic thinking and abstraction, because the explanation of a transaction involves explaining how the complexity and intermediate states are hidden from other users.

Paper 2 emphasizes pattern recognition, algorithmic thinking and evaluation. Students will be presented with tasks that demand the identification of patterns in data sets or the design of efficient algorithms for sorting or searching. For example, an algorithmic question could challenge students to compare the time complexity of two sorting algorithms, requiring not just an understanding of the algorithms themselves but also the ability to evaluate their efficiency and suitability for different data sets. This also extends to the use of abstract data types (ADTs) in solving specific problems. Students may need to decide on the most appropriate ADT (e.g. stack, queue, linked list) for a given scenario, such as implementing a print spooler system. This choice necessitates a deep understanding of abstraction, as students focus on the ADT's operations rather than its implementation. It also requires careful evaluation, as students assess which ADT best meets the problem's requirements.

By using computational thinking skills across both papers, students are better equipped to tackle complex computing challenges. These examples underscore the critical role of computational thinking in the course as a whole.

Computational thinking and IA

The computational thinking process is at the heart of the IA task—the computational solution. The IA criteria embody computational thinking as a process, and the degree to which a student can deploy this type of thinking to a computing problem of their choice will determine their success in the IA.

The skills developed in themes A and B, and assessed in the external assessment components, come together in the IA to be deployed using the computational thinking process. This assessment models the abilities students will require in studies beyond the DP—for example, in computer science university courses and in real-world applications of computer science to academic and industrial problems.

Computational thinking and TOK

Computational thinking is a way of thinking and the key methodology of computer science. As a way to address TOK in the computer science classroom, computational thinking is an excellent starting point. It is not only a way of thinking in computer science but can also be seen in other subjects such as mathematics. Discussions around TOK can consider the similarities and differences of computational thinking in different contexts. It can also help students understand the role that computing can play in other subjects and in solving problems in a wider context beyond academic pursuits.

Building from computational thinking to TOK

Computational thinking extends beyond computer science, serving as a pivotal link to TOK by demonstrating how systematic methodologies can unify and generate knowledge across diverse disciplines. It can be regarded as an interdisciplinary methodology. It shares common ground with various subjects, such as the decomposition of historical analysis, pattern recognition in biology, and mathematical modelling. It has a universal application in understanding complex problems and fostering a holistic approach to knowledge that aligns with TOK.

Computational thinking can also create new knowledge. Through its application in fields like environmental science, it contributes to new insights, such as predicting climate change—exemplifying how technological methods can lead to significant advancements in human understanding, a key area of interest in TOK. The theme of technology in TOK should be explored with computational thinking in mind, in the same way that the natural sciences are considered from the perspective of the scientific method.

Similarly, TOK's exploration of how we know and create new knowledge resonates with the application of computational thinking across healthcare, finance and social sciences, where it integrates and innovates across disciplines. From the impact of machine learning on healthcare diagnostics to algorithmic trading in finance and the use of computational models to understand social phenomena, computational thinking exemplifies the dynamic way we bridge and extend the frontiers of knowledge.

Introduction

This section aims to explore and illustrate approaches to learning and approaches to teaching in the context of DP computer science. This course is distinct in that it is part of the DP where concurrency of learning in different areas of study is emphasized. Connections are therefore encouraged with students' other IB courses, and with both the local and global context.

The IB approaches to learning and approaches to teaching offer a framework of deliberate skills and attitudes underpinning learning and teaching. These approaches aim to support the IB mission and develop skills that enhance students' learning, both during and beyond their DP experience. Connections are therefore made to the learner profile attributes and international-mindedness, and other features at the heart of an IB education, not least a broadly constructivist and student-centred approach, where contextual relevance, concurrency of learning and a connected curriculum are paramount.

The **approaches to learning** framework comprises five skills groups.

- Thinking skills
- Communication skills
- Social skills
- Self-management skills
    - Organizational skills
    - Affective skills
- Research skills

The **approaches to teaching** refer to the six pedagogical principles that underpin IB programmes.

- Teaching based on inquiry
- Teaching focused on conceptual understanding
- Teaching developed in local and global contexts
- Teaching focused on effective teamwork and collaboration
- Teaching designed to remove barriers to learning
- Teaching informed by assessment

IB authorization and evaluation processes require schools to demonstrate implementation, development and review of the approaches to learning and approaches to teaching, as they relate to the *Programme standards and practices*.

This section suggests how to develop different approaches to learning and approaches to teaching—for example, sharing ways to help foster conceptual understanding. There are also ideas for including international-mindedness in the course and suggestions linking topics to real-world contexts. These aim to enhance students' interest and help them see the relevance of their learning to current global challenges.

Approaches to learning

The approaches to learning framework aims to develop skills in students that support their learning throughout the DP/Career-related Programme (CP) and beyond. The skills of computer science are both technical and personal. The personal skills are focused on the ability to collaborate, to think imaginatively, to engage in the process of computational thinking and to think critically. These personal skills are the same as those in the approaches to learning; they can be seen as essential to developing knowledge of computer science and the ability to solve problems using computational thinking.

The challenge for teachers is to both find activities that develop these skills and recognize where specific tasks to develop practical skills in computer science and content delivery can also be used to develop the approaches to learning. The following sections include examples of opportunities to develop approaches to learning in the DP computer science classroom.

Thinking skills

Two overarching frameworks in the DP computer science course are computational thinking, and the specific computer science skills of algorithmic thinking and programming. The role of these computer science skills in computational thinking is important, and should be brought out in classroom activities and content delivery. It is therefore recommended that teachers emphasize computational thinking in computer science and its role as the organizing concept of the course.

Examples

- A classroom activity to support the development of computational thinking could involve giving students complex data sets and asking them to develop a simple recommendation system. For example, a data set that details food ordering on an online platform over a period of time.
    - Students are provided with the raw data. They are then asked to look for patterns in the data: pattern recognition.
    - Students must then identify the specific steps needed to create accurate recommendations: decomposition.
    - They then devise an algorithmic approach to building the system: algorithmic thinking.
- When considering the purpose, benefits and limitations of modern digital infrastructures, students could be presented with a scenario of a smart traffic light.
    - Students must consider the different perspectives of multiple users such as drivers, emergency vehicles and cyclists: problem specification.
    - Students then identify how the system has functions to solve their problems: algorithmic thinking.
    - They could then identify the successes and limitations of the current solution to the scenario: testing and evaluation.

Communication skills

Communication in computer science takes many forms, and different parts of the process of computational thinking require different forms of communication. These include plain language, charts, diagrams, computer code and comments. Mastering these different forms of communication is essential for students to share their ideas with others successfully and understand ideas that have been shared with them.

Communication is also the ability to share and discuss ideas in verbal and non-written forms. This includes the development of the skills needed to make the video for IA. Video is an effective and efficient communications medium, demonstrating the functionality of a specific product and showing that it has been tested.

As teachers develop ideas with students, it is recommended that they help them clearly see how communication skills are essential to developing skills in different elements of the course content and their role in the computational thinking process.

Examples

- During database design (A3.2) students will be expected to produce diagrams to communicate the design of their database solution. This will include conceptual, logical and physical schemas. Examples are entity relationship diagrams (ERDs) and communicating the design of entities in third normal form (3NF).
- Students will be developing their own code throughout the course. It is important they understand that their code should be readable by others—in the real world, coding is completed in teams. To aid readability and communication in their code, students should be using appropriate names for variables and methods, and adding comments to aid understanding.
- As part of the IA task students are required to communicate the design of their solution in a clear, succinct way. One way of doing this is to develop clear, diagrammatic communication such as system diagrams, flow charts, pseudocode and user interface (UI) diagrams.

Social skills

Students’ social skills operate at various levels—among peers, with other members of the school community, between the student body and the local community, and between students and the much wider global community. Social skills in computer science are focused on collaborating and working with others to find solutions. As such, learner profile attributes that are particularly important when developing social skills and collaborating with others include:

- **inquirers**: learning with others
- **principled**: respect for the ideas and rights of others
- **open-minded**: seeking and being open to other points of view and ideas
- **caring**: empathy, compassion and respect for fellow collaborators.

The collaborative sciences project is an excellent opportunity to focus on certain social skills, given its primary focus on collaboration. In addition, the project’s emphasis on the United Nations Sustainable Development Goals fosters awareness of global issues, and therefore an awareness of the situations of others.

Example

- Students work in groups to develop a solution to a real-world problem. For example, they could be asked to develop a database to manage the inventory for the technology department.
    - As a group, they analyse the current system and work to understand the needs of the inventory database.
    - They work together to identify the different tasks the database needs to manage.
    - They collaborate to normalize and design the database.
    - Together, they construct the SQL commands that will make the database functional.

Self-management skills

Self-management skills are classified into two categories.

- Organizational skills

- Affective skills

Effective organization can encourage independence in students. The ability to work independently is essential in computer science, given its focus on inquiry and problem-solving. Solving complex problems requires problem decomposition and careful planning, and the skills of managing and organizing time, tasks and resources are essential to this process.

Affective skills relate to traits such as state of mind, self-motivation and resilience. This skill set is linked to the learner profile attribute of risk-takers, which seeks to equip students with the determination and forethought needed to face uncertainty, challenges and change (International Baccalaureate, 2020). Complex problems are generally challenging to solve; affective skills are therefore vital to becoming an effective computer scientist.

Examples

- Organizational skills include problem decomposition, planning, time-management skills and selecting the correct resources. Classroom activities that could support this might involve the scaffolding of effective organizational skills. For example, teachers could model the decomposition of tasks in classroom problems, gradually removing their support and empowering students to do this themselves.
- Students could be required to complete mini projects throughout the course (e.g. “drop everything and code” time). For example, students are given problems and then expected to plan their time effectively to complete the task. As students become more adept at programming, they could then be encouraged to select the necessary technical tools and decide whether to develop the solution in (for example) a specific programming language, database or web-based solution.
- Affective skills allow students to build resilience. One way of doing this could be a practical work notebook. Students should be encouraged to document their work, especially the setbacks they have had and how they have learned from them and developed. Teachers can give space for this by allowing students to complete projects in class time that enable them to revisit the design and development stage iteratively, creating refined solutions to problems.

Research skills

Research plays a role in computer science, such as research undertaken in computer science libraries for existing solutions or researching domain knowledge to understand how to specify a problem in its context.

Students need to know how to find and access current and relevant established knowledge in computer science as part of their research, and clearly recognize the work of others. Students also need to know how to conduct relevant research of domain knowledge so that they can effectively specify a problem in a context with which they are unfamiliar.

Examples

- Students will find research useful when they are looking for ways to develop programming solutions. An example could be referring to the application programming interface (API) documentation for Java and Python. It is expected that students will want to develop useful and complex programs. Therefore, researching the associated API documentation will enable them to understand the tools available when developing solutions in these coding languages.
- Another example is identifying a problem in context. Students should spend time researching the context thoroughly. This may include data analysis, documentation analysis, reviewing help files and frequently asked questions, and observing and assessing the context of the problem.

Approaches to teaching

Inquiry

Inquiry has several goals, one of which is to model the inquiry process so that students are prepared to ask questions for themselves when beginning the IA—the computational solution. The computational solution is a central element of the DP computer science course that requires the student to identify a problem of their own choosing and develop a software solution for it, using the computational thinking process.

Inquiry-based approaches to learning and approaches to teaching involve a high degree of engagement and interaction to develop natural curiosity in students. The approaches can take a variety of forms that differ in their degree of teacher guidance. Banchi and Bell (2008) propose four levels of inquiry.

1. As part of **confirmation inquiry**, the question, process and outcome of the inquiry are provided by the teacher.
2. During **structured inquiry**, teacher guidance begins to be withdrawn.
3. During **guided inquiry**, teacher guidance continues to be withdrawn.
4. In **open inquiry**, students determine the question, process and outcome.

Other forms of inquiry-based learning and teaching include experiential learning (Kolb, 1984) and problem-based learning (Boud, Feletti, 1997).

The inquiry process in computer science requires several steps that are focused on computational thinking. These steps require students to specify a problem, decompose the problem into sub-problems, and, once a solution has been achieved, reconsider the solution as part of an iterative process of improvement.

The following are examples of how inquiry-based approaches use different levels of support so that scaffolding can help students to develop these inquiry skills for themselves. This inquiry approach is recommended both as part of the acquisition of knowledge from the syllabus, and the development of algorithmic thinking and programming.

Examples

Confirmation inquiry

Students perform a teacher-directed inquiry to confirm an outcome.

- Learning to use an integrated development environment (IDE), especially to quickly write, compile, run and test simple programs. An IDE is a software application that provides a comprehensive set of tools for software development. IDEs are designed to make it easier to write, debug and test code, and typically include a code editor, a debugger and a range of other features such as syntax highlighting, code completion and integration of version control.
- Using examples of variables, simple data structures, control flow, selection structures and exception handling, students solve progressively more difficult directed problem sets, where they check their answers against those of the teacher.

Structured inquiry

Students determine the outcome themselves; the aim and procedure are provided by the teacher.

- As students learn about functions, teachers can show students how to calculate the area of a square. Students can then choose other geometric shapes and determine their areas.
- “Fizz buzz” is a classic computer science programming problem in which a program iterates through an ordered set of numbers from 0 to 100. If a number is divisible by three the word “fizz” is printed, if a

number is divisible by five, “buzz” is printed, and if a number is divisible by both, “fizzbuzz” is printed. Students can decide to look for other divisible combinations once they have solved “fizzbuzz”.

Guided inquiry

Students conduct an inquiry that addresses a question provided by the teacher, using a procedure of their choice.

- “How can you efficiently store and retrieve student information for a school database that is volatile?”
    - Students will explore different data structures (e.g. lists, dictionaries, tuples, sets) to determine the most efficient way to store and retrieve student information, including names, grades and classes.
- “How can you implement an efficient system for navigating and retrieving points of interest within a game map using spatial data structures?”
    - Students will explore and implement spatial data structures to create an efficient navigation system for a game map. This system will manage and retrieve points of interest like treasures, enemies and landmarks, based on the player's location or search criteria.

Open inquiry

Students determine the question, process and analysis in several possible ways.

- They write their own factual, conceptual and debatable questions in relation to a topic.
- They undertake the collaborative sciences project.
- They decompose a real-world problem and solve it.

Experiential learning and problem-based learning

Outside the classroom, students engage in inquiry opportunities with a problem they have chosen for themselves. In effect this is the IA computational solution. However, the skills required for students to tackle the computational solution should be developed throughout the course. It is recommended that this form of inquiry is experienced by students at least once before they undertake the computational solution.

Conceptual understanding

The overarching organizing concept in the DP computer science course is the computational thinking process. However, there are many other concepts within the course. In IB programmes, conceptual understanding is defined as understanding that connects factual, procedural and metacognitive knowledge. This understanding develops from a process by which students consciously organize connections between prior and new knowledge into networks, then further develop or reconfigure those networks. This is a non-linear, ongoing process throughout which understandings evolve and misconceptions are identified and dispelled. In DP computer science, these interconnections are explored through the linking questions. Note that the linking questions in the *Computer science guide* are not exhaustive, and students and teachers are invited to write their own.

Teaching for conceptual understanding is important because conceptual understanding enables students to be aware and critical of their own knowledge and understandings. Teaching approaches that promote conceptual understanding include classification, generalization, representation, internalization, concepts-in-use, and near and far transfer. A conceptual approach fosters the organization of knowledge into networks, which can evolve as students acquire new understandings. These can be broad concepts that help to integrate knowledge across disciplines, or narrower, subject-specific concepts that help to organize and link disciplinary understandings.

The following are examples of activities that develop conceptual understanding—to link different areas of the syllabus, to link the syllabus to understandings beyond computer science, and to show how these conceptual understandings relate to the computational thinking process.

Examples

- When revising several units, students write their own linking questions, share them and attempt to answer each other's questions. For example, in "A1 Computer fundamentals" we encounter the linking question:

  "What role does task-scheduling in an operating system play in managing network traffic and requests?"

  In this case, teachers could employ a visible thinking routine—using the "Connect, Extend, Challenge" approach to demonstrate a conceptual connection to the linking question (Project Zero, 2015).

  - Connect: Have students connected the concept of task-scheduling to their existing knowledge about how computers manage multiple processes simultaneously?
  - Extend: The teacher encourages the students to extend their thinking by considering how task scheduling impacts not just the CPU's efficiency, but also the management of network traffic.
  - Challenge: The students are asked how these concepts would apply in a real-world scenario, such as in the operation of a popular online service during peak usage times.
- Break it down, build it up: this type of approach especially supports computational thinking.

  Purpose

  - To help students practise and internalize the process of decomposition by breaking down complex concepts, problems or systems into their constituent parts, and understanding how these parts work together to form the whole.

  Break it down

  - Identify the whole: Begin with a complex problem, concept, or system the students are trying to understand or solve. This could be a software project, a challenging algorithm, or a computer system.
  - List the components: Students identify all the components or elements that make up the whole. This encourages detailed examination and analysis.
  - Describe functions: For each component identified, students describe its function or role within the whole. This helps them understand the purpose and necessity of each part.

  Find the interconnections

  - Map relationships: Students map out how the identified components interact with one another. This could be through data flow diagrams, unified modelling language (UML) diagrams, or simple connection maps.
  - Analyse interactions: Students discuss or write about how these components work together to achieve the system's overall functionality.

  Build it up (rebuild the whole)

  - Synthesize: Students explain or demonstrate how the components, once combined, form the complex whole they started with. This could involve summarizing the process, creating a simplified model or diagram of the system, or even reconstructing the problem with a clearer understanding.
  - Reflect: Students reflect on the process of decomposition and how it helped them understand the complex whole better. They might answer questions like: "How did breaking down the problem change your understanding of it?" or "What was the most challenging part of the system to understand, and why?"
  - Apply: Encourage students to apply this routine to a new or related problem, reinforcing the skill of decomposition as a general problem-solving tool.

Local and global contexts

The primary goal of computer science is to solve problems, and these often come from the real world. The most familiar context for students is the local one—the world immediately around them at home, at school, and in their local communities and social networks. Inquiry-based learning can use these contexts as a starting point for problems that students solve using computer science. Local contexts should also be related to global contexts where possible.

Examples

- At regular intervals throughout the course, students discuss computing-related news items, considering:
    - relevant theoretical concepts
    - links to topics that have been studied in the computer science class
    - examples of ethical, environmental, economic, cultural and social impacts.
- Invite industry professionals, school alumni or students' relatives who work in computing-related careers to present to the class. This will be especially effective if the speaker can help students work through a real-world problem in their field, so that students learn to think like a computer scientist.
- Does your school have a network manager? Ask them to come to class to field questions from the students.
- At regular intervals throughout the course, highlighting and discussing a specific area within computing: computer science, computer engineering, software engineering, information systems, information technology.

Effective teamwork and collaboration

Teamwork and collaboration is a common and highly effective way of solving problems using computer science. This work can occur face to face or online. The development of solutions to problems in the real world of computer science is rarely an individual effort but more commonly a collaborative one involving computer scientists and other experts with specialist domain knowledge. The use of suitable digital tools to enhance sharing, collaboration and teamwork in computer science is part of the skill set of a computer scientist.

Collaboration and teamwork in the DP computer science course should be developed through activities that model effective teamwork, both face to face and online. For students to benefit fully from collaborative tasks, behavioural expectations and the nature of effective group work need to be made explicit. Teachers must therefore clarify the difference between acceptable collaboration and collusion. Examples of collaboration and teamwork from the real world of computer science can help students understand its place in the development of solutions. The collaborative sciences project can also be a focus for collaboration between computer science students and students with different domain knowledge.

Examples

1. "Pair programming" is a well-known software development technique in which two programmers work together at one workstation. One, the "driver", writes the initial code, while the other, the "observer" or "navigator", reviews each line of code as it is typed in. The two programmers switch roles frequently.
2. Students should be encouraged to **observe** changes made in popular GitHub repositories. For example:

   https://github.com/trending

   https://github.com/moodle/moodle/commits/main/
3. Students should ask effective technical questions of each other, and in so doing, support each other as teammates. Teachers should be careful that high-ability students do not do all the work for their peers. A good technical question is precise, outlining the specific goals to be achieved, evidencing the effort required to solve it and includes specific error messages.

Removing barriers to learning

IB approaches to teaching aim to help students set challenging and appropriate personal learning goals. Differentiation to remove barriers to learning relates to four interconnected principles. The central principle is to affirm students' identities and build self-esteem. Surrounding this are the principles of valuing prior knowledge, scaffolding learning and then extending learning.

In IB programmes, language is recognized as having a vital role in learning and teaching, permeating almost all aspects of a learning environment. All IB teachers are therefore regarded as language teachers. Many terms used when explaining computer science concepts are highly technical and have meanings that are not usually considered in everyday contexts. Teachers should be mindful of this and make definitions explicit for students, helping them to understand how context may influence the meaning of a word or phrase.

Students may well begin the course with different levels of skills in algorithmic thinking and computer programming. Teachers need to be mindful of this and ensure they have a clear understanding of each student's skill level when they begin the course. This will be important in informing the approaches to teaching, the nature of differentiation and planning the course to cater to all students' prior abilities and experiences.

Examples

Some activities that can support the removal of barriers to learning include the following.

- Affirming students' identities and building self-esteem (e.g. students construct their own glossaries of computer science meanings through a diagram-based rather than language-based approach).
- Valuing prior learning by identifying prior knowledge through quizzes, introductory activities and activate learning through linking questions.
- Scaffolded learning is best realized by asking students to self-select sets of problems of varying difficulty.
- Extending learning could involve applying concepts in unfamiliar contexts and exploring interdisciplinary links.

Assessment

Assessment provides valuable information that supports learning and teaching and therefore should be continuous throughout the course. Formative assessment feeds back into the teaching process, providing information that facilitates the synchronization of learning and teaching. It is a key two-way process that can be used to improve learning, teaching and further assessment. Both teachers and students' peers provide students with feedback on how to consolidate their understanding and move forward. Meanwhile, students provide teachers with feedback on their misconceptions, knowledge gaps, levels of understanding and levels of engagement, all of which inform the teacher's subsequent decisions.

Throughout the DP and CP, students work towards demonstrating their abilities in the formal assessment of the course (e.g. examination papers, IA and projects such as the collaborative sciences project). Assessment of learning, including formative assessment, should therefore be aligned with the requirements of the course assessments and grade descriptors, with appropriate feedback and reporting. Throughout the course, activities should aim to provide opportunities for students to demonstrate their skills in computer science, both technical and personal. The aim of activities should be to produce outcomes where these skills are visible and can be formatively assessed, whether informally or formally with structured feedback. The technical skills and techniques of computer science include algorithmic thinking and programming. The personal skills include the approaches to learning, collaboration and inquiry.

In computer science, formative assessment will also provide opportunities for the students to develop the skills and techniques of computational thinking required to undertake the computational solution. Students should also be exposed to questions of the style and type they will encounter in external examinations.

Examples

- Debug deliberately broken code: in simpler iterations, this formative activity assesses to what extent students understand basic syntax and control structures. In more complex iterations, it invites students to ask "What is this code supposed to do?" and debug and enhance the code more thoughtfully.
- Students consider and discuss the advantages and disadvantages of specific technical solutions.
  - Technical decisions usually involve more than one potentially acceptable solution (e.g. which data structure to use to solve a particular problem). Most technical solutions are a trade-off between multiple factors such as performance, complexity, scalability, maintainability and resource usage.
  - Choosing a data structure might involve balancing between quick access times and memory efficiency. For example, a hash table might offer fast lookup times but can be memory intensive compared to a linked list, which could have slower access times but uses memory more efficiently for sequential access patterns.
- Self-assessed activity: design and evaluate a simple application.
  - Students conceptualize, design and develop a simple application—e.g. a task manager, a basic calculator, or a personal diary—to solve a computational problem.
  - Students then evaluate their solution against a set of success criteria they develop themselves, reflecting on their design and development process in the context of computational thinking and IA skills.
- Peer assessment of a UML diagram.
  - A student designs a system using UML diagrams that adequately represents software architecture or a process.
  - A different student then interprets the diagram to explain the system's functionality, structure and the interaction between components, without additional input from the diagram's creator.
- Self-assessment and peer-based assessment using rubrics and markschemes.

Upskilling as a teacher of the course

Upskilling as a teacher is essential for delivering high-quality education and staying relevant in the rapidly evolving field of computer science. Upskilling can be necessary both for the overarching concepts required by the course, but also for more specific topics that may be new to the teacher. For example, some elements of programming in the course are now topics that are no longer optional and must be taught.

The following is a list of upskilling suggestions for DP computer science teachers.

Activities and resources

Professional development courses

PD courses offered by the IB can provide nuanced answers to many questions teachers will have, and also offer ideas for upskilling after the course has been completed. As well as receiving training and information during the course, teachers may make contact with other teachers who can support them with additional resources and further ideas for upskilling. PD is one of the most effective ways of building a network of peers: to which teachers can contribute and from which they will tangibly benefit.

Programme Communities

IB Programme Communities offer opportunities for IB teachers to support each other with information, ideas and resources. They are a place to build and nurture teacher networks by both sharing ideas and asking for help. The next three recommendations can often be found in such communities, exemplifying how beneficial they can be.

Online courses and resources

Computer science is one of the best-served subjects in terms of the availability of online courses and other resources such as online videos. For example, massive open online courses (MOOCs) are a good way to develop in-depth expertise on a subject, while video resources on other online platforms can also be a valuable support for specific points, even if the video is short. Recommendations and support from other teachers on Programme Communities are a helpful way to find the courses to meet individual teachers' upskilling needs.

Articles and books

The DP computer science course is inherently future facing, given continuous advances in the field, so it is important to be aware of developments that are relevant to the course. Teachers should keep abreast of developments they see in news stories or in articles from computer science publications. These can be particularly helpful for the case study and could also support students in coming up with ideas for the IA task.

Books are also useful upskilling resources, especially for detailed programming skills. In this regard, teachers are likely to find that their Programme Communities are an ideal place to look for and share recommendations with their peers.

Practice and application

One of the best ways for teachers to upskill is to design and develop their own activities. This can be time consuming, so once again, sharing the burden with other teachers by collaborating in online Programme Communities can be an effective and efficient way of developing context-specific resources. Using these exercises and activities in the classroom is first and foremost to the students' benefit, of course, but it will also help to build teacher confidence in the specific area being practised.

Further suggestions can be found in Resources for upskilling (PDF).

Areas of focus

Computational thinking

Computer science teachers will likely have an understanding of computational thinking, but it is important that they are clear about its meaning as defined for this course in the *Computer science guide*. It is therefore recommended that teachers spend time thinking about and researching those aspects of computational thinking outlined in the guide that they are less familiar with.

Puzzles, coding challenges and everyday problems will give students the opportunity to practise (for example) decomposition, pattern recognition, abstraction and algorithmic thinking. This will support their development but also develop the teacher's confidence in computational thinking.

Machine learning

It is vital that teachers build a foundational understanding of machine learning. It is also important that they consider strategies for delivering machine learning concepts in the classroom. Undertaking machine leaning projects in class is recommended, from simple ones such as building a linear regression model to more complex deep learning projects.

There are a wide range of MOOCs covering machine learning available, and online video channels that can be very helpful for upskilling. There are also online resources specifically designed for teaching machine learning. It is important that teachers share both their expertise and resources to make the most of this new and innovative field.

Programming: Java or Python

It will be necessary for some teachers to strengthen their programming skills in either Python or Java, depending on which language they intend to use to teach the course. The focus of upskilling for the DP course will be OOP, algorithms and data structures.

Object-oriented programming

Teachers may need to deepen their understanding of OOP principles (encapsulation, inheritance, polymorphism and abstraction) and how to teach these effectively and apply them in Python or Java. To this end there are many online courses that focus specifically on OOP; the most effective courses will include hands-on projects to apply learning.

There are a number of publications that support each of the languages. For Python programming, the official Python documentation provides a strong foundation. It is also recommended to apply OOP principles by working on projects. Simple projects like a library management system are a good starting point, from which it is possible to gradually increase complexity. GitHub can be used to document and version projects, allowing for reflection and iterative improvement.

Algorithms

Another area to upskill is the ability to solve problems using algorithms and to teach algorithmic thinking. It is recommended that the focus is on sorting, searching and problem-solving strategies.

Courses that offer problem sets for practical application are recommended. There are also a number of online platforms available for practising algorithmic problems, starting with easy problems and progressively tackling more complex challenges.

It is important that teachers understand Big O notation and how to analyse the efficiency of algorithms. This knowledge is crucial for teaching students to write optimized code.

Data structures

Teachers may need to upskill to master the use and implementation of fundamental data structures—such as arrays, linked lists, stacks, queues, trees and graphs—and understand their application in solving computational problems.

In Python or Java, data structures can be implemented to aid understanding of the underlying mechanics and efficiency of each structure.

Online courses can provide both theoretical background and practical coding exercises. They can also help with solving coding challenges that require the use of specific data structures, explaining when and why to use certain data structures over others.

International-mindedness

The IB aims to develop "inquiring, knowledgeable and caring young people who help to create a better and more peaceful world through education that builds intercultural understanding and respect". International-mindedness recognizes similarities while also affirming differences between communities, peoples and nations. Knowledge and understanding of similarities allow for the construction of common foundations, while recognition and affirmation of differences encourage valuing and celebrating diversity.

With this in mind, teachers of DP computer science should provide opportunities for students to foster international-mindedness within the context of the course, underpinned by a focus on global engagement. Global engagement represents a commitment to address humanity's greatest challenges in the classroom and beyond. Such challenges may relate to the environment, development, conflict, rights and cooperation.

One of the aims of the course is to design and implement solutions to local and global problems using the process of computational thinking. DP computer science students and teachers are therefore expected to explore local and global issues related to the content of the syllabus. There is a close connection between international-mindedness and the IB learner profile attributes, which underpin, and are central to, understanding what it means to be internationally minded.

Along with their exposure to international-mindedness elsewhere in the DP, students should be prepared to be successful global citizens of the future.

Integrating international-mindedness

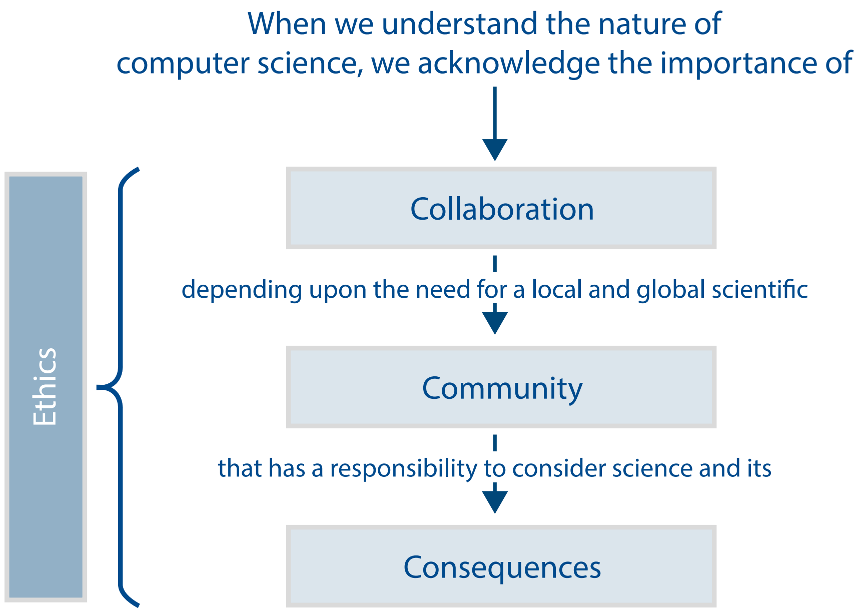
Teachers are not expected to address international-mindedness as a stand-alone topic, but instead to integrate it in teaching the subject. This TSM therefore aims to help teachers address international-mindedness in their delivery of the course. The ideas and suggestions that follow are neither mandatory nor exhaustive, and teachers are encouraged to generate alternative approaches and share ideas as they plan and deliver the content.

The "Computer science and international-mindedness" section of the *Computer science guide* highlights the international nature of computer science, and is a good starting point to consider the role of a computer scientist in a global community.

For practical purposes, it might be useful to break down the term "international-mindedness" into specific concepts that are particularly relevant in the context of science: collaboration, community, consequences and ethics. Note also that a consideration of ethics can provide valuable overlap with the TOK course.

Figure 1

Integrating computer science with other concepts


When we understand the nature of computer science, we acknowledge the importance of
Ethics
Collaboration
depending upon the need for a local and global scientific
Community
that has a responsibility to consider science and its
Consequences

Scaffolding questions for developing international-mindedness

The following table lists some scaffolding questions that could be used to develop international-mindedness. They can also be applied to specific real-life contexts.

Concept	Scaffolding questions for developing international-mindedness
Collaboration	• Why is collaboration necessary? • What type of collaboration is effective in computer science? • What does collaboration in computer science look like?
Community	• Who is involved in the computer science community? • What structures and organizations are found in the computer science community?
Consequences	• Which areas of society might be impacted by new knowledge in computer science? • Does every society respond similarly to developments in computer science?
Ethics	• What responsibilities do individual computer scientists have in carrying out their own research? • What are the responsibilities of online communities, technology companies and open-source movements? • Should developments in computer science be subject to ethical constraints? • How can ethical constraints be put into practice by society for developments in computer science?

Exploring international-mindedness

Real-world context	Connections to international-mindedness
Developments in machine learning to produce arts	The ethical considerations of using machines to develop artworks: How can we ensure all voices are represented and not just the majority? The consequences of using machines to develop artworks, which may hinder human creativity in the future.
Developments in machine learning to solve problems	Which community's voice is the most powerful when it comes to developing AI machines, and is this community democratic? What are the consequences of some voices going unheard in the development of machine learning? What collaborative boards exist to monitor development?
Developments in open-source software movement	Open-source software is a clear example of wide collaboration in computer science. It demonstrates the computer science community building knowledge together. Programmers should credit the original developers and continue to develop the community further.
Developments in network security	The communities of computer scientists working on counter measures in network security. Communities developing software that could be used to perform network attacks on others. Consequences of network attacks on individuals and larger communities.
Developments in programming software	IDEs are created to enable easier collaboration within programming teams. Collaboration on complex programming software builds and develops diverse communities. A community of capable and effective programmers is developed, with broader perspectives.
Developments in data warehousing	Data warehousing enables querying and analysis of data to provide generalized insights from a central repository of information. Data analytics can be used to build communities by identifying connections between people from diverse experiences, locations and cultures. The consequences in society of storing and performing analytics on stored data and ethical considerations of storing all data about people.

Examples

The following examples are general structures that could be used to integrate international-mindedness in lessons or parts of lessons.

Research and discussion

One activity that may foster international-mindedness in the classroom is researching the role of online analytical processing (OLAP) and data mining for business intelligence. Students can research the different analytics completed on the data and identify uses for the analytics. They could consider the ethics of the analysis being performed on the data, as well as the implications of using the analytics to make decisions in different scenarios. During their research, students should investigate a range of diverse perspectives. The discussion should focus on the inclusion of these perspectives when making ethical decisions about the uses of data mining.

Mini-presentations

A typical presentation could focus on the processes involved in developing a complex problem. Students present their ideas on investigating complex problems to their peers, focusing on the diversity of experiences and various approaches. This presentation enables students to understand the problem from different perspectives and points of view, and to consider the ethical implications of investigating their problem from diverse perspectives.

Debate

The ethics of integrating technologies into everyday life is a good starting point for debate. The teacher can formulate a question using a suggested scenario; students research the scenario and formulate a response based on the evidence. Students then collaboratively consider the implications of their research and findings from at least two different perspectives. These perspectives are then used as the starting point for a debate in the classroom.

Introduction

The DP computer science course does not prescribe a specific sequence for covering the course concepts and topics. Teachers are completely free to plan their own route through the course according to their circumstances and the needs of their students. This section therefore offers ideas for potential routes and contexts that may help teachers to develop their own course outlines.

It is expected that the development of students' skills in computational thinking, algorithmic thinking and programming are integrated in the delivery of course content. This section includes suggestions for planning the delivery of course content effectively and coherently, supporting conceptual understanding and developing students' skills in computer science.

Pathways through the course

Teaching in context

Context 1: Making decisions

Students could explore the following topics within the context “Making decisions”.

- A1.2.3 Describe the purpose and use of logic gates.
- A1.2.4 Construct and analyse truth tables.
- A1.2.5 Construct logic diagrams.
- B1.1.3 Explain how applying computational thinking to fundamental concepts is used to approach and solve problems in computer science.
- B2.3.3 Construct programs that utilize looping structures to perform repeated actions.

Using a given problem, students could inquire into the different decisions being made and how these would be represented at different levels of abstraction. A sequence of lessons for making decisions may look similar to the following.

Lesson 1	Introduction to the idea of decisions within programs and the decisions students are asking the computer applications to make on a regular basis. This could then lead into a discussion about simple problems that require decisions to be made. For example, what to do at the weekend based on different variables.
Lesson 2	Identify the different variables that need to be considered when making a given decision. Develop an algorithm for solving the given problem. For example, if it is raining and cold at the weekend, then go to the cinema. Students could then code this in either Python or Java.
Lesson 3	Introduction to the idea of logic gates and how they can be used to represent expressions, such as the decisions identified in the previous lessons. For example (A AND B). Students then construct the truth table for their selection statements.
Lesson 4	Students use the truth tables to construct logic diagrams for given problems.

Context 2: Automated vehicles

Students could explore the following topics within the context “Automated vehicles”.

- A1.3.7 Explain the use of control systems in a range of real-world applications. (HL only)
- A2.1.2 Describe the purpose, benefits and limitations of modern digital infrastructures.
  (Specifically, the role of edge computing in automated vehicles)
- A4.1.1 Describe the types of machine learning and their applications in the real world.
- A4.4.2 Discuss ethical aspects of the increasing integration of computer technologies into daily life.

Using automated vehicles as a context, students can inquire into the use of control systems and digital infrastructures in the real world. A sequence of lessons for automated vehicles may look similar to the following.

Lesson 1	Introduction to autonomous vehicles and their key components. What inputs are the vehicles working with and what processing needs to happen? Describing the issues of

	processing fully happening onboard the vehicle and introducing the need for cutting edge technology to maximize the efficiency of the vehicle.
Lesson 2	Identify the learning that is happening onboard the vehicle and identify the way in which the learning is happening.
Lesson 3	Discuss the ethics of having automated vehicles being tested in society. Specific examples of successes and drawbacks can be pulled from the public domain.

Developing a scope and sequence

A scope and sequence can be developed in different ways. One possible scope and sequence is given in the following section and focuses on an inquiry-based approach to the course. The example units are built around specific inquiries that students undertake to develop knowledge, understanding and skills in computer science.

Inquiry-based process topic sequence

The following is one example of a sequence of units of work based on the topics in the syllabus, i.e. it is not the only possible way of grouping and sequencing the content. This sequence of units is for SL only. A summary table is given and the focus is on inquiry.

Unit	Title	Linked subtopics	Rationale
1	Computer fundamentals	A1.1, A1.2, A1.3	An understanding of the internal workings of the computer underpins all knowledge in computer science. Students can tie together the different aspects of the course with this first unit.
2	Introduction to computational thinking and programming	B1.1, B2.1, B2.2, B2.3, B2.4, B2.5	To solve complex programming problems, students need to understand the different techniques employed in computational thinking. An introduction to programming also develops computational thinking skills in context.
3	Object-oriented programming	B3.1	The concept of classes helps students build on their coding and computational thinking skills to construct increasingly robust programs.
4	Databases	A3.1, A3.2, A3.3	Once students understand computational thinking within the OOP context, these skills can be developed within databases. This enables students to further understand the nature of computational thinking.
5	Networks	A2.1, A2.2, A2.3, A2.4	Networks are fundamental in understanding the developments that have taken place in computer science. These include open-source developments, machine learning and remote working. An awareness of how interconnectivity works will enable students to understand the importance of network architecture.

Unit	Title	Linked subtopics	Rationale
6	Machine learning	A4.1, A4.4	Machine learning is now fundamental to computer science. Developing knowledge of machine learning will help students harness its power to solve increasingly complex problems.

The focus of inquiry for each unit is outlined below.

Note: The unit inquiries are constructed from topics in the syllabus, and the guiding questions can be used as an initial basis of inquiry. Teachers are also encouraged to construct their own guiding questions appropriate to the unit inquiry. In the examples below, this example teacher's own guiding questions are broken down into more specific sub-questions, also developed by the teacher.

Unit inquiries

Unit 1: Computer fundamentals

Guiding question: How does the functionality of the computer system enable us to develop solutions?

A1.1 Computer hardware and operation

- How do the components of the computer system work together to maximize efficiency?
- What components of computer systems allow users to run resource-heavy processes such as video games, simulations and AI?
- Why is it important to consider the memory limitations of a computer system when trying to run resource-heavy processes?

A1.2 Data representation and computer logic

- How is data represented in physical memory within a system?
- How can we model real-world decisions within a computer system?
- How can we model the circuits that are making the decisions within the system?

A1.3 Operating systems and control systems

- What is the role of the operating system?
- What features of the operating system allow the computer to complete multiple tasks?
- How does the computer system deal with unexpected events?

Unit 2: Introduction to computational thinking and programming

Guiding question: How can we use computational thinking to develop a computational solution to a real-world problem?

B1.1 Approaches to computational thinking

- How can a complex problem be described in a problem specification?
- What concepts are used in computational thinking to solve problems?
- How can solutions to problems be depicted using computational thinking?

B2.1 Programming fundamentals

- How is data stored within computer programs?
- How can data stored in variables be manipulated?
- What techniques are available to programmers to test their programs?

B2.2 Data structures

- What data structures can be utilized when developing a solution?
- How can programmers make use of fundamental operations within their program?
- How can 1D and 2D dynamic lists be implemented to develop solutions?

B2.3 Programming constructs

- How are decisions made within a program?
- How can code instructions be sequenced to solve problems?
- Why is it important to develop code using modularization?

B2.4 Programming algorithms

- What measures are there for determining the efficiency of an algorithm?
- How can data be manipulated to find specific information?
- What sorting options are there for data?

B2.5 File processing

- How can data be saved permanently in a program?

Unit 3: Object-oriented programming

Guiding question: How does the design of a program reflect the function of the program?

B3.1 Fundamentals of OOP for a single class

- How can classes be used to represent the real world?
- How can classes be used to create complex solutions?
- To what extent are classes essential to modularization?

Unit 4: Databases

Guiding question: Can databases offer an alternative solution to complex problems?

A3.1 Database fundamentals

- What are databases used for?
- How can databases be used to solve complex problems?
- How are databases used to represent real-world entities?

A3.2 Database design

- How can computational thinking be modelled in the database paradigm?
- What techniques are available to develop an efficient database?
- How can data from complex problems be stored in databases?

A3.3 Database programming

- What techniques are there to manipulate databases?
- How can data from complex problems be manipulated and used in databases?
- When would you use a database to solve a real-world problem?

Unit 5: Networks

Guiding questions: What are the concepts that underpin how networks operate?

A2.1 Network fundamentals

- What are the characteristics of a network?
- How can networks enhance solutions to complex problems?
- What hardware and protocols are required for networks to function?

A2.2 Network architecture

- How are networks used in real-world solutions?
- What networking models are available when developing solutions?
- What are the real-world applications of networking models?

A2.3 Data transmissions

- How is data transmitted over a network?

- What are the limitations of data transmission and how may this affect the performance of networked solutions?

A2.4 Network security

- What threats exist in networked solutions and what countermeasures are available to combat them?

Unit 6: Machine learning

Guiding question: What principles should be considered to ensure machine learning models produce accurate results ethically?

A4.1 Machine learning fundamentals

- What are the different approaches to machine learning?
- What are the real-world applications of each approach to machine learning?
- What hardware is required to support machine learning?

A4.4 Ethical considerations

- What challenges are there within machine learning in terms of accountability, algorithmic fairness, bias, consent, environmental impact, privacy, security and transparency? How does this affect society?
- When developing a machine learning solution, what biases need to be identified and how can they be avoided?
- Why is it important to regularly revisit ethical guidelines?

Maximizing opportunities for early programming experience

In DP computer science, students often present a range of programming abilities. It is advisable to bring in programming early in the course so that students can develop their programming skills over the full length of the course. Developing a bank of programming challenges will help with differentiating the skill levels of students, and allow the teacher to support students in the most appropriate way during class activities.

The following are some different scenarios to consider.

What if some students have little or no programming experience?

The SL course does not require students to have any prior knowledge of; meanwhile some HL students may only have limited programming experience, especially of Java or Python. This means there are some students with no experience of programming at all, and others with only limited experience. One way this can be addressed is to bring in “offline” computer science skills very early on in teaching. There are many resources available that enable students to practise computational thinking prior to programming. For example, in the computer fundamentals unit, students could be introduced to binary and hexadecimal coding using a code-breaking challenge. The computational thinking elements students will need to be able to access this are pattern recognition and decomposition. Engaging with tasks like these can enable students to prepare for coding.

What if there is a mix of SL and HL students with different levels of programming skills?

If there is a mixed-level class, assessing prior knowledge will allow the teacher to understand the skill level of each student. This data can differentiate students into three groups: beginner, novice and intermediate. Students in the beginner group will be supported through tasks with additional explanations and modelling from the teacher. In the novice group, there will be some support from the teacher but less direct scaffolding. Intermediate students can be given minimal support; from the beginning, they could also be given suggestions for extensions that would enhance the software. One way of developing a collaborative environment is to encourage intermediate-level students to support their beginner and novice peers on tasks that develop programming skill.

How does the programming and algorithmic thinking fit with the scope and sequence?

Algorithmic thinking should permeate the course from the beginning and tasks given to students should have an inquiry element. By incorporating inquiry from the start, students will begin to understand the algorithmic thinking process. No matter which unit the teacher uses to begin with, linking the programming and content to the computational thinking process will help to develop the skills students need to complete the course.

Computer science applications

Students who are capable of inquiry-based learning in computer science—particularly in context—will develop strong skills and a good conceptual understanding of the subject. This can be a challenging approach for students and teachers alike, but the rewards make the learning more meaningful and more likely to be applied to solve problems beyond the computer science classroom. One concrete effect of this is that students will be far better prepared for the IA task: the computational solution.

The following techniques can be used by teachers to help students fully understand the computational thinking process. This understanding is essential to success in the IA task.

Complex problem generation	Students should be encouraged to bring problems to class. Examples could include the following questions. • How do students get enrolled in extracurricular activities? • How are bus routes decided? • How do the local ticket barriers work? Teachers can use these as contexts for teaching. Examples include using: • extracurricular activities to demonstrate a sorting algorithm • bus routes to show how algorithms may function in machine learning • ticket barriers to discuss how different aspects of a system integrate with each other.
Complex problem identification	Throughout the course, teachers could identify points of further investigation that highlight to students opportunities that could be taken in their IA task.
Modelled solutions	It is important that teachers take every opportunity to model the computational thinking process. They can set up sessions for mini-projects, where students inquire into developing a mini-solution to a small problem once they have acquired the necessary knowledge. Examples could include the following. • How could we build a small network to share our computer science notes effectively? • What algorithm could we use in machine learning to discover our shared areas for growth? • How could we build a program to help us decide future careers? • How can we make use of a database to make a computer science questionbank?

Unit planners

Example unit planners

All DP teachers are required to engage in explicit planning. However, the IB does not prescribe a particular format of unit planner that teachers should use. Nevertheless, the process of planning may be supported by using or adapting one or more of the example DP unit planners developed by the IB for DP teachers. These planners are not intended to mandate or restrict what teachers can or cannot do; instead they are designed to give teachers practical ideas for teaching the course content. These examples are intended to help teachers to reflect on their own planning, but they are not "model" plans, nor are their formats and approach to content prescriptions of how unit planning should be undertaken.

Example DP unit planners are provided in the publication *Approaches to teaching and learning in the Diploma Programme*.

Example unit planner
Unit planner 1—Object-oriented programming (SL and HL) (Word)

Skills-based activities

Practical skills

As well as covering the course content in themes A and B, students need to develop their skills in algorithmic thinking and programming. It is recommended that students' development of these skills is integrated with delivery of the course content. However, there are situations where specific stand-alone activities could also be especially effective; some examples are given in the following sections.

Algorithmic thinking

Activity 1

Consider the game tic-tac-toe (noughts and crosses). How would a computer track players' positions in a game? Students should consider lists and arrays as ways of representing a game board.

Activity 2

Consider a game of rock-paper-scissors. How might a computer represent choices and scores? Students would be expected to understand variables, perhaps using lists or arrays.

Activity 3

Consider searching for a personal ID number in a country. If this country has 50 million ID numbers that are not sorted, what approach do students think would be the fastest way to find the number? Students would be expected to think about iterating through a data structure.

Programming

Java examples

Activity 1: Decision-making

Course content: A1.2.2, A1.2.3, B2.1.1, B2.3.2

An activity that can be used to demonstrate decision-making is coding some simple decision-making questions. Examples could be: What should I do at the weekend? How should I revise for my mathematics examinations? There are several different ways to approach this, allowing teachers to model the computational thinking process.

Activity 2: String manipulation

Course content: A1.2.2, B2.1.1, B1.2

An activity that could be used to demonstrate string manipulation is to create a simple Caesar's cipher. Students then use the cipher to explore the difference between a string and a char. This task enables students to make use of several of the string class methods, and also reinforces the need for different variable types.

Activity 3: Using classes

Course content: B2.2.2, B3.1

An activity to demonstrate classes could be to create a simple storage program that involves searching and sorting. For example, a movie database that allows the user to add a movie they have watched, give it a rating and name the platform on which they watched it. Once built, users could interrogate the system to find movies according to their rating and see which platforms they are available on.

Python examples

Problem sets are an effective method to teach and practise programming. Here are some problems that can be solved with Python.

- Find the middle character in a string of any size, accounting for even/odd numbers of characters.
- Write a program that prints the very first item in a list and the very last item in a list.
- Rotate an array 90 degrees

[[1,2,3],

[4,5,6],

[7, 8, 9]]

so that it returns:

[[7, 4, 1],

[8, 5, 2],

[9, 6, 3]]

- A common mechanism in computer games is to simulate a dice roll. Write a function that rolls dice for you and returns the result. For example:
    - roll_dice(d6,3) <-- would roll a six-sided die three times and return the sum of the three rolls as the result
    - roll_dice(d12,1) <-- would roll a 12-sided die once and return the result
    - roll_dice(d20,4) <-- would roll a 20-sided die 4 times and return the sum of the 4 rolls as the result.

Inquiry skills

The fundamental, IB-recommended approach to teaching is for this to be based on inquiry. The following are some example activities that focus on inquiry and specific situations where students can see the computational thinking process in action.

Inquiry 1: Developing a virtual pet simulator

This inquiry focuses on computational thinking to solve a problem in computer science. Students will design and program a simple virtual pet (e.g. a dog, cat, imaginary creature) that users can interact with through feeding, playing and health monitoring. This project aims to introduce basic programming concepts, problem-solving and the application of computational thinking skills in a simulated environment.

Computational thinking focus

Decomposition	Breaking down the virtual pet’s needs and behaviours into manageable programming tasks (e.g. hunger level, happiness level, health status)
Pattern recognition	Identifying similarities in how different actions affect the pet’s state (e.g. both feeding and playing increase happiness but may have different effects on health)
Abstraction	Creating generalized representations of pet actions and states, focusing on the logic behind pet care rather than detailed, realistic simulations
Algorithmic thinking	Developing algorithms to simulate the pet’s responses to user actions and the passage of time

Solution design

The steps involved in developing the virtual pet are as follows.

Introducing the virtual pet concept

1. Explain the concept of a virtual pet and how it can simulate aspects of caring for a real pet through programming.

2. Discuss the goals of the virtual pet simulator, such as keeping the pet healthy and happy.

Designing the virtual pet

3. Students list the key features and behaviours of their virtual pet, such as hunger, health and mood.

4. Guide students in defining abstract metrics to represent these features (e.g. numerical values for hunger levels).

Developing basic interactions

5. Instruct students to program basic interactions such as feeding the pet, which decreases hunger, or playing with the pet, which increases happiness. Use conditional statements to change the pet's state based on these interactions.

6. Implement simple commands that users can input to interact with their pet (e.g. "feed", "play").

7. Implement consequences over time.

- Introduce the concept of time passing in the game, affecting the pet's needs even without user interaction. For example, hunger increases over time, requiring regular feeding.
- Use loops and time delays to simulate the passage of time and its impact on the pet.

Testing and debugging

8. Encourage students to test their simulator, checking for logical errors or unexpected behaviour in the pet's responses.

9. Guide them through debugging any issues that arise, emphasizing the importance of testing in the development process.

Reflection and discussion

10. Discuss how computational thinking skills were applied in designing and programming the virtual pet simulator.

11. Reflect on the challenges encountered during the project and how they were addressed.

Inquiry 2: Building a simple encryption–decryption tool

This inquiry focuses on computational thinking to solve a problem in the real world. It will guide students through the process of creating a tool that can encrypt and decrypt text using a simple cipher (e.g. a Caesar or substitution cipher). The project aims to reinforce computational thinking skills by applying them to a real-world computing problem: securing information.

Computational thinking focus

Decomposition	Breaking down the task into smaller, more manageable parts—designing the cipher, coding the encryption algorithm, coding the decryption algorithm and developing a UI
Pattern recognition	Identifying patterns in the encryption process that can be systematically applied for decryption
Abstraction	Focusing on the general principles of encryption and decryption without getting bogged down in the specific details of a specific language or system
Algorithmic thinking	Developing a step-by-step solution for the encryption and decryption process

Solution design

The steps involved in building a simple encryption–decryption tool are as follows.

Introduce encryption and decryption

1. Begin with a brief overview of encryption and decryption, explaining their importance in securing digital information. Link to relevant course content on data security and cryptography.

Explore cipher techniques

2. Introduce a simple cipher technique—such as the Caesar cipher—explaining how it shifts letters by a certain number down the alphabet for encryption, and reverses the process for decryption.

3. Discuss the concept of keys in cryptography, and their role in encryption and decryption.

Design the tool

4. Students outline the components of their tool, including the input (plaintext), the encryption algorithm, the decryption algorithm and the output (ciphertext).

5. Students define the process abstractly first, such as "shift each letter in the plaintext by three positions to encrypt" before moving on to coding.

Develop the encryption and decryption algorithms

6. Guide students through coding the encryption and decryption algorithms in the programming language of choice. This step reinforces algorithmic thinking and implementation skills.

7. Implement error handling for edge cases, such as letters near the end of the alphabet or non-alphabetical characters.

Implement a user interface

8. Students create a simple UI that allows users to enter text, choose to encrypt or decrypt it, and display the resulting text. This can be a command line interface (CLI) or a basic GUI if students are ready for more advanced programming concepts.

Testing and debugging

9. Encourage students to test their tool with various inputs to ensure that both the encryption and decryption processes work correctly. This step involves critical evaluation and debugging skills.

Reflect and discuss

10. Conclude the activity with a discussion on how computational thinking skills were applied throughout the project. Reflect on the challenges encountered and how they were overcome.

Inquiry 3: Analysing DNA sequences with Python

This inquiry focuses on the application of computer science in another subject. The objective is to guide students through the development of a Python program to analyse DNA sequences, identifying characteristics such as GC-content, frequency of nucleotide patterns, or even simulating DNA mutations. This project aims to apply computational thinking and programming skills to a biological context, enhancing students' understanding of both genetics and computer science.

Computational thinking focus

Decomposition	Breaking down the DNA analysis task into smaller parts, such as reading a DNA sequence, calculating GC content and identifying specific nucleotide patterns
Pattern recognition	Identifying common patterns within DNA sequences that may have biological significance, such as motifs associated with regulatory regions
Abstraction	Developing functions or methods that perform specific tasks related to DNA analysis, making the program modular and reusable for different DNA sequences
Algorithmic thinking	Creating algorithms to process and analyse the DNA sequence data effectively, applying logic to biological data analysis

Solution design

The steps involved in analysing DNA sequences with Python are as follows.

Introduction to DNA sequencing

1. Begin with a brief overview of the structure of DNA, its function and the importance of sequence analysis in biology, linking to relevant topics in the biology curriculum, such as genetics, evolution and molecular biology.

Understanding DNA data

2. Discuss the representation of DNA sequences as strings of nucleotides (A, T, C, G) and introduce common analyses performed on DNA sequences, such as GC-content calculation (a measure of stability in genetic sequences) and pattern searching.

Programming fundamentals

3. Review or introduce basic Python programming concepts necessary for the project, including string manipulation, loops, conditionals and functions.

Developing the DNA analysis tool

Decomposition

4. Assign specific tasks for the tool, such as reading sequence data from a file, calculating GC-content, and searching for specific nucleotide sequences or patterns.

Algorithmic thinking

5. Guide students through developing algorithms for each task. For example, to calculate GC-content, count the number of Gs and Cs in the sequence and divide by the total length of the sequence.

Implementation

6. Students implement their algorithms in Python, writing code to analyse sample DNA sequences provided as part of the activity or obtained from public databases.

Testing and debugging

7. Students test their program with known DNA sequences to ensure accuracy in calculations and pattern recognition. Discuss the importance of accurate data analysis in biology and the implications of errors.

Application and discussion

8. Students then use their tool to answer a real-world biological question, such as analysing the DNA sequence of a gene associated with a specific trait or disease. Discuss the findings and the role of computational tools in biological research.

9. Reflect on the interdisciplinary nature of the activity, emphasizing how computer science skills can enhance understanding and discovery in biology.

Learning skills

It is recommended that learning activities are developed that support students in developing their computational thinking. As is the case for other specific skills, such activities will ideally be integrated into the learning and teaching of the course content.

The following are some example activities that focus on specific elements of computational thinking.

Activity 1: Problem specification and decomposition

Example 1: Designing a classroom resource management system

Work individually to design a simple Python-based system to manage classroom resources, such as books, laptops and laboratory equipment. This project encourages students to practise problem specification and decomposition in preparation for coding their solutions.

This example links to programming, OOP, databases and computational thinking.

Computational thinking focus

- Problem specification: Clearly define what the classroom resource management system needs to achieve, including its functionalities, user roles and constraints.

- Decomposition: Break down the overall system into smaller, manageable pieces such as inventory tracking, check-out processes and user-interaction interfaces.

Example 2: Developing a personal finance tracker

Design and implement a simple Java or Python application to track personal finances, including income, expenses and budgeting. This project encourages critical thinking about financial literacy while applying computational thinking skills in problem specification and decomposition.

This example links to programming, OOP, databases and computational thinking.

Computational thinking focus

- Problem specification: Define the functionalities and user requirements of the personal finance tracker, such as tracking different types of expenses, visualizing spending patterns and setting budget goals.
- Decomposition: Break down the overall application into smaller, manageable modules, such as data input, expense categorization, budget comparison and data visualization.

Example 3: Building a sorting algorithm visualizer

Develop a Java or Python application that visually demonstrates the workings of various sorting algorithms (e.g. Bubble Sort, Merge Sort, Quick Sort). This project is designed to deepen students' understanding of sorting algorithms through visual representation, highlighting the efficiency and mechanics of each algorithm.

This example links to programming, OOP, abstract data structures and computational thinking.

Computational thinking focus

- Problem specification: Clearly define the functionalities that the visualizer must have, including the ability to showcase different sorting algorithms, visually display the sorting process and compare the performance of algorithms.
- Decomposition: Break down the project into smaller, manageable components, such as the GUI, algorithm implementation, animation of the sorting process and performance metrics (e.g. complexity, number of comparisons).

Example 4: Developing a course recommendation system with machine learning

Design and implement a simple recommendation system using Java or Python and machine learning libraries (e.g. scikit-learn). This project aims to apply computational thinking in the context of machine learning to solve a practical problem—recommending academic courses to students based on their preferences and academic history.

Computational thinking focus

- Problem specification: Define the system's goals, including accurately predicting and recommending courses that align with student interests and academic needs.
- Decomposition: Break down the project into manageable tasks, such as data collection (gathering student data), feature selection (identifying relevant variables for recommendation), model training (choosing and training the machine learning model) and system evaluation (assessing the accuracy of recommendations).

Activity 2: Abstraction and pattern recognition

Example 1: Designing a classroom resource management system

Develop an abstract model of a classroom resource management system and analyse usage patterns to optimize resource allocation. This exercise emphasizes the skills of abstraction (creating a simplified model of a complex system) and pattern recognition (identifying common trends or behaviours in resource usage).

Computational thinking focus

- Abstraction: Create a simplified representation of the classroom resource management system, focusing on key elements such as resources, users and usage times.
- Pattern recognition: Analyse data to identify common usage patterns, peak times and potential bottlenecks in resource allocation.

Example 2: Developing a personal finance tracker

Design an abstract model of a personal finance tracker and use pattern recognition to analyse spending habits. This activity emphasizes the computational thinking skills of abstraction (simplifying complex financial data into manageable components) and pattern recognition (identifying trends in spending habits).

Computational thinking focus

- Abstraction: Develop a simplified representation of the personal finance tracker, focusing on the essential elements such as income sources, expense categories, savings goals and financial transactions.
- Pattern recognition: Analyse transaction data to identify trends and habits in spending, income and savings, which can inform budget adjustments and financial planning.

Example 3: Building a sorting algorithm visualizer

Develop an abstract model for a sorting algorithm visualizer and use pattern recognition to analyse the efficiency of different sorting algorithms. This activity emphasizes the computational thinking skills of abstraction (simplifying the complex behaviours of sorting algorithms into understandable models) and pattern recognition (identifying efficiency patterns across various data sets).

Computational thinking focus

- Abstraction: Design a simplified representation of the sorting algorithm visualizer, focusing on key elements such as the data set to be sorted, the sorting algorithms themselves, and the visual representation of the sorting process.
- Pattern recognition: Analyse how different sorting algorithms perform under various conditions—such as nearly sorted data, reverse order, random order—to identify which algorithms are most efficient for specific data patterns.

Example 4: Developing a course recommendation system with machine learning

Conceptualize a course recommendation system through abstraction and use pattern recognition to analyse educational data for personalized recommendations. This activity highlights the computational thinking skills of abstraction (simplifying complex systems into understandable models) and pattern recognition (identifying trends and correlations in student data that influence course recommendations).

Computational thinking focus

- Abstraction: Develop a simplified representation of the course recommendation system, focusing on essential elements such as student profiles, course metadata and the recommendation engine (machine learning model).
- Pattern recognition: Analyse student data and course outcomes to identify patterns that can guide the development of the recommendation algorithm (e.g. student interests, academic performance, course popularity).

Activity 3: Debugging and testing functionality

This is an activity that focuses on the steps of debugging and testing functionality. Link these to the course content where possible.

For each of examples 1 to 4 in activity 2, the overall activities of testing and debugging are similar. These are:

1. developing a structured approach to ensure the software functions correctly and efficiently manages classroom resources such as books, laptops and laboratory equipment

2. reproducing bugs—use information from failed tests to reproduce bugs
3. using debugging tools—utilize the features of an IDE or debugging tools to step through code and identify the source of errors
4. logging analysis—review application logs for error messages or anomalies that occurred during testing
5. functional testing—ensure all features work as intended, such as resource checkouts, returns, inventory tracking and user management
6. usability testing—assess the UI for intuitiveness and ease of use by teachers and students.

Activity 4: Evaluation

Example 1: Designing a classroom resource management system

Functionality evaluation

- Assess whether all system features work as intended.
- Verify that the system supports all specified requirements, such as tracking resource availability, managing check-outs and returns, and generating reports.

Usability evaluation

- Determine how intuitive and user-friendly the system interface is for its users.
- Conduct user testing sessions to gather feedback on the system's ease of use, the understandability of its features, and the overall user experience. This often involves observing users as they complete typical tasks within the system and collecting their subjective feedback.

Performance evaluation

- Measure the system's response time, its reliability under load, and its ability to handle concurrent users without significant performance degradation.
- This may involve stress testing the system to ensure it remains stable and responsive during peak usage times.

Example 2: Developing a personal finance tracker

Feature completeness

- Verify that all advertised features—such as budget creation, expense tracking, financial goal setting, report generation—are implemented and functioning as expected.

Accuracy

- Test the accuracy of financial calculations, including expense categorizations, budget summaries and forecasting projections.

Example 3: Building a sorting algorithm visualizer

Functionality evaluation

- Algorithm coverage: Ensure that the visualizer includes a diverse range of sorting algorithms (e.g. Bubble Sort, Quick Sort, Merge Sort) and accurately demonstrates their sorting processes.

Visual representation

- Assess the clarity and correctness of the visual representations for each algorithm, ensuring they effectively illustrate the algorithm's mechanics.

Interactivity

- Evaluate the system's interactivity, such as allowing users to control the speed of the visualization, pausing/resuming the sorting process, and input of custom data sets for sorting.

Example 4: Developing a course recommendation system with machine learning

Accuracy and relevance evaluation

- Prediction accuracy: Measure the accuracy of the recommendation system in suggesting courses that align with student interests and academic goals. This can be quantified using metrics such as precision, recall and F1 score.
- Relevance of recommendations: Gather user feedback on the relevance of recommended courses to assess whether the system is effectively personalizing suggestions based on individual user profiles.

Usability evaluation

- Ease of use: Determine how user-friendly the system interface is, ensuring that students can easily navigate, understand and utilize the recommendation features.
- Feedback mechanism: Evaluate the effectiveness of any feedback mechanisms that allow users to refine their preferences or provide feedback on recommendations, enhancing the system's learning and adaptation over time.

Personalization evaluation

- Depth of personalization: Assess the system's ability to tailor recommendations to the unique academic and career goals of each student, considering a wide range of factors including past course performance, interests and future aspirations.
- Adaptability: Test how well the system adapts to changing user preferences and academic progress over time, ensuring that recommendations remain relevant and supportive of the student's educational journey.

Paper 1

Paper 1 tests knowledge and understanding of theme A: Concepts of computer science. This is the case for both the SL and HL versions of this paper.

The paper includes two sections: section A with questions on syllabus topics A1, A2, A3 and A4, and section B with questions related to the case study.

More detail on the structure of the external examination papers can be found in the DP *Computer science guide*.

Theme A

Theme A covers four key areas in computer science. The first topic, "A1 Computer fundamentals", delves into computer hardware, data representation, operating systems, and for HL students only, translation. The second topic, "A2 Networks", explores the basics of network architecture, data transmission and network security. "A3 Databases" is the third section, focusing on database fundamentals, design, programming, and for HL students only, alternative databases and data warehouses. The final section, "A4 Machine learning", introduces machine learning basics and ethical considerations, with additional HL-only subtopics on data preprocessing and machine learning approaches.

Across all these areas, the learning methods outlined as follows emphasize a blend of theoretical knowledge and practical application, fostering both conceptual understanding and real-world skills. Interactive and project-based learning, combined with case studies and real-world applications, are key to effectively acquiring the knowledge and skills outlined in theme A.

The knowledge and skills required for theme A can be broken down into its four key topics.

A1 Computer fundamentals

Analytical skills

- Analyse how hardware components interact with each other.

The learning approach integrates hands-on hardware assembly for practical understanding of component roles, utilizes hardware simulation software to analyse visual interactions, and involves real-world case studies of hardware successes and failures to grasp their applications.

Logical reasoning

- Construct and interpret a series of Boolean logical operations.

A possible learning approach includes the use of online interactive simulations for creating and experimenting with logic circuits, alongside exercises in number system conversion. These offer students an engaging and practical approach to understanding these key concepts.

Systems thinking

- Understand how operating systems manage hardware and software resources.

One learning approach is to use comparative and simulation-based exercises of various operating systems in computers, smartphones and control systems, highlighting their unique resource management strategies.

Critical thinking

- Analyse the process of converting human-readable code into a form that a computer can execute (HL only).

A learning approach could incorporate a practical example of a simple program in a high-level language with the corresponding assembly language and machine code, enhancing students' understanding of the translation process from high-level code to machine-executable format.

A2 Networks

Critical thinking

- Evaluate the purpose, benefits and limitations of modern digital infrastructures such as the internet, cloud computing and mobile networks.

One learning approach could foster critical analysis through discussions and research projects focusing on the impact of these technologies in various sectors, such as finance (e.g. cryptocurrency blockchain), urban planning (e.g. smart traffic lights) and education (e.g. school networks).

Systems thinking

- Understand the concepts and applications of network segmentation for performance and security, including techniques to reduce congestion and manage resources efficiently.

An effective learning approach is to conduct practical exercises in network segmentation, preferably using network simulation tools. Include project-based learning where students must segment a network to optimize performance and security, considering real-world scenarios such as corporate or educational networks.

Technical proficiency

- Compare types of media for data transmission (fibre optic, twisted pair, wireless), considering their advantages and disadvantages.

A recommended learning approach is to create a hands-on laboratory where students experiment with different transmission media, observing factors such as bandwidth, range and interference. Supplement with case studies that highlight real-world applications and choices of transmission media in different contexts.

Problem-solving

- Identify and address common network vulnerabilities such as distributed denial of service (DDoS) attacks, malware, man-in-the-middle (MitM) attacks and SQL injection (HL only).

A possible learning approach is to conduct scenario-based exercises where students must diagnose and propose solutions to simulated network attacks. Include the study of real-world incidents to understand the application of countermeasures in practical situations.

A3 Databases

Technical proficiency

- Understand the technical aspects of relational databases, including key features such as concurrency control, data integrity and transaction processing.

One recommended learning approach would be to provide students with a partially completed relational database using either Java or Python with SQL. Students undertake tasks to create and manipulate the relational databases, focusing on implementing features like primary and foreign keys, while discussing data consistency and integrity.

Systems thinking

- Apply the principles of normalization in database design, including the different normal forms (1NF, 2NF, 3NF) to common database scenarios, such as medical, police, education and commercial databases.

One learning approach is to use exercises that guide students through the normalization process, starting from unnormalized tables and progressing through the various normal forms. Scenarios are included to illustrate common normalization issues such as data duplication and dependency concerns.

Analytical thinking

- Construct complex SQL queries to retrieve and manipulate data from multiple tables, including the use of joins.

A suggested learning approach is to conduct hands-on exercises where students develop SQL queries involving joins, relational operators and pattern matching. Challenges are presented that require students to order and filter data using SQL commands, emphasizing the nuances of syntax across different database systems.

Data analysis

- Understand the role of online analytical processing (OLAP) and data mining techniques in business intelligence.

A learning approach is to implement practical exercises or simulations where students apply data mining techniques like classification, clustering and regression, on large data sets. Encourage the integration of these skills with machine learning concepts to extract meaningful insights for business intelligence.

A4 Machine learning

Synthesis

- Understanding different types of machine learning (e.g. deep learning, reinforcement learning, supervised learning, transfer learning, unsupervised learning) and how they are applied to real-world applications.

One learning approach is to assign a project where students research and analyse a specific real-world scenario, identifying which machine learning approach is best suited to that situation. Students present their findings, explaining their rationale.

Technological literacy

- Develop proficiency in using data preprocessing tools and techniques.

One learning approach would be to provide students with a spreadsheet containing unformatted, anomalous and badly coded data to clean up. This could be extended by incorporating the use of data analysis software or programming languages (e.g. Python, R) in laboratory sessions, where students get hands-on experience with data preprocessing tools. This will help them become familiar with real-world applications and build their technological literacy in data science.

Decision-making

- Comprehend the importance of model selection and comparison in machine learning.

One learning approach would be to facilitate discussions on case studies where students must choose appropriate machine learning models for different types of data and problems. They are encouraged to explore how different algorithms perform under varying data characteristics and problem complexities.

Ethical decision-making

- Integrate ethical considerations into technological understanding.

One learning approach is role-playing exercises where students must navigate scenarios involving emerging technologies (e.g. quantum computing, augmented reality, pervasive AI). The importance of reassessing ethical guidelines as technology evolves is emphasized, such as the implications for privacy, equity and individual rights.

Case study

As students develop theoretical knowledge, they are expected to put it into practice through a hands-on case study, providing a shared framework for discussions and ideas.

The purpose of the case study is to familiarize students with recent developments and emerging technologies in computer science. The case study should be seen as a stimulus for further research. If this research is not done, students will find it difficult to answer questions on the case study in section B of paper 1.

Teachers should approach the case study as a dynamic component of the curriculum, rather than as a discrete unit. This enables the integration of case study topics with other areas of the syllabus, reinforcing understanding through application. While in-depth expertise is not necessary, a foundational grasp of the relevant technologies is crucial for guiding discussions and linking theoretical concepts with practical examples.

Strategies for incorporating the case study

When exploring the case study, teachers can facilitate the process by identifying and discussing the key technologies and challenges involved. These focal points often correlate with the syllabus, allowing for a coherent learning experience. It is important that students not only comprehend the technologies at a superficial level as presented in the case study documentation, but also engage with the underlying principles and hurdles associated with the technologies.

Teachers should ensure that students understand the scenario and the key concepts of the case study. Researching the definition of the key terms is a starting point that can then be reinforced and practised with quizzes and peer explanations. Students can peer-teach areas of the research and describe similar scenarios they have uncovered while asking "what if …?" questions. To foster a deeper understanding, teachers can use various pedagogical strategies—such as orchestrating group research projects, leading topical discussions, guiding debates on mock scenarios and arranging group or individual presentations. These activities encourage collaborative learning and critical thinking, with mind mapping serving as an excellent tool for visualizing the connections between the case study and syllabus content. The goal is to ensure that students are well prepared for the examination by equipping them with a robust understanding of the case study, its technologies and associated challenges.

Research requirements and skills development

In terms of the research needed for the case study, it is essential to aim for depth over breadth. This means focusing on the types of research that will allow students to gain a profound understanding of the core technologies and challenges mentioned in the case study. The research should be varied in type, including academic articles, industry publications, white papers and case histories to provide a well-rounded perspective. The quantity of research should be enough to allow students to form a comprehensive understanding without it becoming overwhelming.

Students should be able to:

- apply their theoretical knowledge to practical and real-world scenarios
- analyse and evaluate the implications of the technologies in the case study
- synthesize information from various sources to form a coherent understanding of the case study
- develop arguments and solutions based on their research findings.

The skills students should hone include critical thinking, problem-solving, research, collaboration and communication. These are not only vital for tackling the case study but are also transferable skills that will benefit them in their further studies and professional lives.

Discussions, essays and guided analyses should focus not just on the technical details, but also the:

- ethical implications of the technology in the case study
- socioeconomic effects and considerations
- sustainability of the technology and its long-term viability

- security concerns associated with the technology.

This process will prepare students not just for the examination, but for thoughtful engagement with future technologies and scenarios they may face.

Additional requirement for HL students

HL students will encounter an additional complex question in paper 1, section B that necessitates not only a comprehensive and detailed response but also demonstration of higher-order thinking skills. This question will typically require HL students to evaluate the application of a technology in a certain situation, justify their reasoning, discuss the technology's broader implications, or explain to what extent certain statements or scenarios hold true. This kind of question is designed to assess their depth of understanding and their ability to engage in critical analysis and synthesis. It is an opportunity for HL students to showcase their ability to articulate nuanced insights and to apply their knowledge to complex, multifaceted problems, distinguishing their level of engagement from that required at SL.

In such scenarios it is important for students to be comfortable with giving a balanced evaluation. Even if one possible solution may seem more or most suitable, it is extremely rare that answers are clear cut. Therefore, students should practise objectively presenting their arguments from a neutral point of view and justify any conclusions or recommendations they make with specific research they have carried out, rather than merely offering their opinion.

Paper 2

Paper 2 assesses knowledge and understanding of theme B: Computational thinking and problem-solving. While theme A focuses on conceptual understanding, theme B is a more skills-based paper. In computer science the two key technical skills are algorithmic thinking and programming. These skills are essential in using the computational thinking process successfully to solve problems.

Paper 2 tests algorithmic thinking, programming with abstract data structures, and OOP. Note that OOP is studied at greater depth at HL.

Algorithmic thinking

Algorithmic thinking is a foundational skill in computer science, crucial for problem-solving and effective programming. In the context of DP computer science, the development of algorithmic thinking skills is essential, not only for success in external assessment but also for students' future endeavours in the field.

Paper 2 gives students the opportunity to demonstrate their computational thinking skills in answers to questions that do not require knowledge of any particular programming language. Paper 2 will always present one question of this type, where coding is not necessary.

In this question students may be presented with flow charts representing algorithms and asked to analyse or explain their functionality. Alternatively, students may need to trace the execution of given algorithms, identifying the values of variables at each step and predicting the output. A specific problem may be presented, for which students are required to devise solutions without necessarily implementing them in code. These exercises should be within all students' reach, irrespective of their programming expertise, as long as the exercises recommended previously or other suitable exercises are regularly practised in class.

Developing algorithmic skills

One approach to cultivating algorithmic thinking skills in students is through exercises and activities that do not necessarily involve programming. By focusing on the logic and structure of algorithms, students can grasp fundamental concepts without the complexity of coding syntax.

The following table indicates some strategies for developing algorithmic thinking.

Flow chart interpretation	Students can practise interpreting and creating flow charts to represent algorithms.
Algorithm tracing	Tracing existing algorithms allows students to step through the instructions manually, tracking the flow of control and the values of variables at each step.
Problem-solving activities	Engage students in problem-solving tasks that require algorithmic thinking, such as solving puzzles, challenges involving logical reasoning, or real-world problem scenarios.

When engaging in these types of activities, students should be given the opportunity to practise each of the computational thinking skills: decomposition, pattern recognition, abstraction, algorithmic thinking, testing and evaluation.

Programming

Students have the option to take paper 2 in either Python or Java. Both versions are equally challenging, using the same contexts and covering the same topics. Although the questions are very similar across both versions of the paper, there are slight adjustments to fit the specific programming language. Students should choose one of the languages and answer all questions on the paper using the language specified. Questions may cover a variety of tasks that assess different skills, including:

- program reading—analysing existing code to understand its functionality and purpose
- code modification—making changes or additions to given code to achieve specific objectives
- program evaluation—assessing the correctness, efficiency and usability of code solutions
- program generation—creating code to solve specific problems as set out in the examination question.

In the examination, especially for Python questions, students may need to write code for common tasks (e.g. sorting, date calculations) without using pre-existing libraries or functions. This tests their ability to code basic functionalities manually, even though knowing how to use libraries is still useful. If a question does not allow a specific function or library this will be clearly stated.

The examination will also test knowledge of OOP concepts. It should be noted that HL students are expected to understand this topic in greater depth and that the questions will therefore be more challenging.

Additional requirement for HL students

In addition to the SL material, HL students are required to be familiar with all the additional data structures in "B.4 Abstract data types" and OOP concepts related to multiple classes as specified in B3.2. This applies both to the Java and Python papers, though it is expected that the questions in each paper may vary slightly as there are fundamental differences between the two languages in these areas.

Developing programming skills

To develop students' proficiencies in programming skills, teachers are encouraged to engage students in various activities encompassing algorithmic thinking, programming knowledge, problem-solving, code writing, error diagnosis, optimization and analysis. The following suggested activities equip students with the necessary competencies to navigate programming challenges effectively and create robust software solutions.

Algorithmic thinking

Essential for programming, algorithmic thinking focuses on breaking down problems into sequential, logical steps for efficient problem-solving.

- Design algorithms for organising a bookshelf, focusing on sorting and categorizing techniques.

Programming knowledge

Understanding the syntax, semantics and libraries of a programming language to write code effectively.

- Explore and summarize different data structures in Python, creating sample code to demonstrate their use.

Problem-solving skills

Identifying and solving challenges in coding tasks.

- Solve a puzzle by creating a program that finds the shortest path in a maze, focusing on logic and strategy.

Code writing and implementation

The practical application of programming concepts to create functional software or scripts.

- Implement a simple calculator app that performs basic arithmetic operations, focusing on clean code practices and UI design.

Programming error diagnosis

Identifying and fixing bugs in code to ensure correct program execution.

- Exchange code snippets with intentional errors for peers to find and correct, enhancing debugging skills.

Code optimization

Enhancing code efficiency and performance without altering its functionality.

- Optimize an existing code snippet to reduce execution time or memory usage, documenting the changes and impacts.

Programming analysis

Evaluating code for readability, efficiency and adherence to best practices.

- Conduct code reviews in small groups, critiquing and suggesting improvements for better performance and maintainability.

Teacher and student responsibilities

This section outlines the actions and responsibilities of both teachers and students when working on the computational solution. The teacher is the mentor who provides guidance and support to students during the process, so that students can demonstrate the necessary inquiry skills described in the sections of the *Computer science guide*: "The approaches to learning framework", "Approaches to teaching" and "Skills in the study of computer science".

Teachers are encouraged to accompany and guide students on this journey, allowing them to make their own decisions with confidence, replicating the processes undertaken by computer scientists. Topics that have some personal significance for students will encourage them to take more innovative approaches.

Teacher responsibilities

General responsibilities

To ensure ongoing authenticity of the student work, teachers must provide guidance and support as students develop their computational solution. This may include:

- discussing with students their initial ideas for a problem scenario and the computational context for a solution
- being aware of the student's research and data collection
- discussing the system being developed and techniques used to enable a solution
- discussing the student's testing strategy
- scrutinizing the draft and final versions of the work.

Specific responsibilities

While facilitating the computational solution, teachers have the following responsibilities.

Prior to student development of a problem scenario, teachers must:

- explain the aims and requirements of the computational solution, ensuring that the criteria are well understood by students
- ensure that students have developed the required skills for the computational solution
- encourage students to explore ideas and ask questions about solutions that relate to topics studied during the course
- ensure that approximately 35 hours are allocated for the design and development of the computational solution.

During student development of a problem scenario and planning, teachers must:

- ensure that the proposed computational solution requires a software solution to a problem with sufficient complexity
- counsel students on whether the projected solution is suitable in terms of time needed and resources required
- ensure that students have developed appropriate success criteria for their solution
- ensure that each student develops their own computational solution.

During initial student development of the solution, teachers must:

- guide students in developing appropriate representations of system overviews
- ensure that students are aware of how to present their algorithmic thinking in system overviews

- ensure that students consider testing strategies when developing their system.

During the development of the solution, teachers must:

- supervise the programming of the solution being developed by students; if students are programming outside the classroom, sufficient steps should be established to ensure authenticity of student work
- monitor student progress in choosing techniques to implement their algorithms
- where necessary, show students how to choose programming techniques, or guide them in the steps needed to carry out specific tasks—such as creating a functional GUI; file/database connection/interrogation; data manipulation read/operations/output; use of libraries; implementing standard algorithms for sorting, searching, using recursion; implementing different data structures; working with complex logic conditions.

  However, the teacher should not choose a technique for the students, nor do any programming for them.

During the evaluation of the solution, teachers must:

- ensure that students test their solution against the success criteria
- ensure that students make a video—of no more than five minutes in length—demonstrating the functionality of the solution and giving examples of the testing strategy.

During the write-up of the solution, teachers must:

- read and comment on students' draft computational solutions

  The teacher's comments should aim to help students identify issues and shortfalls according to the assessment criteria, but they must not directly offer solutions.
- remind students of the requirements of the computational solution and that they should cover each of the five assessment criteria in separate sections in the documentation they submit
- ensure that students include the full source code in an appendix to their computational solution
- ensure that students include a video—of no more than five minutes in length—using a common format (MP4, wmv, avi); a longer video should not be sped up to fit the maximum length
- ensure that students understand concepts related to academic integrity (e.g. authenticity and intellectual property; any use of AI should be clearly acknowledged)—see "Appendix 6" of the IB *Academic integrity policy* for further guidance
- ensure that the internally assessed work is entirely that of each student, and that any information incorporated from external sources is appropriately cited and acknowledged.

Student responsibilities

General responsibilities

The work students submit must be their own. Students must understand and actively apply concepts related to academic integrity, such as authenticity, respect for intellectual property, and citing and referencing according to accepted systems. This includes instances where AI tools are used, such as to refine code. If students have researched an existing solution, this must be clearly referenced and cited in a bibliography. Students must include the full source code of their solution in an appendix, and excerpts of code in the documentation should reference the appended full source code.

Specific responsibilities

Prior to the development of a problem scenario, students must:

- read and understand the assessment criteria.

During development of a problem scenario, students must:

- choose an appropriate problem scenario that is open to a software solution
- consult with the teacher to ensure that the proposed problem scenario will require a solution that has sufficient complexity

- choose a suitable context in computing for the computational solution
- ensure that success criteria are clearly defined in terms of the problem scenario and computational context.

During the planning and development of the solution, students must:

- consult with the teacher regarding the viability and feasibility of their plan for the solution
- consult with the teacher to ensure they are clearly demonstrating their algorithmic thinking in an appropriate representation
- consult with the teacher to check the techniques being chosen to implement their algorithms
- develop a clear testing strategy related to the success criteria.

During the report writing phase, students must:

- prepare the documentation of their solution and ensure that it includes sections to address each of the criteria in turn
- produce a video, of not more than five minutes, demonstrating the functionality of the product and examples of the testing strategy
- prepare an appendix that includes the full source code of their product
- acknowledge all sources of information with adequate references, including their use of AI; any source code in the body of their solution should reference the full source code in the appendix
- present a draft of their computational solution to the teacher
- consider the teacher’s recommendations on the draft of their computational solution before submitting the final version
- confirm that the computational solution submitted is their authentic work and constitutes the final version of that work.

Scheduling and planning the internal assessment

As stated in the *Computer science guide*, 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". Each school is free to decide at what point in the course it will be most appropriate to begin work on the computational solution. This will usually be when students have developed sufficient skills and subject content knowledge.

Factors to consider in scheduling and planning the computational solution include:

- identifying the point in the course where students have sufficiently developed the relevant practical skills of algorithmic thinking and programming to begin work
- deciding when to schedule the 35 hours to be allocated to the work
- taking into consideration the number of students being assessed, all of whom will need guidance and supervision
- coordinating with teachers of other DP subjects, to avoid excessive student workload
- taking account of school calendars, local regulations and other DP assessment activities
- noting the IB submission deadline indicated in *Diploma Programme Assessment procedures* (updated annually).

Student preparedness

Over the course of their studies, students' preparedness for IA is developed through multifaceted learning opportunities.

Developing approaches to learning skills in computer science

As noted in the *Computer science guide*, the skills of computer science are both personal skills and practical skills. The personal skills required for the computational solution are the approaches to learning experiences described in the "Skills and development" table in "The approaches to learning framework" section of the guide, and in the "Approaches to learning" section earlier in this TSM. Through these experiences, the approaches to learning skills—thinking, communication, social, research and self-management—can be practised and developed before the computational solution is attempted.

Developing practical skills in computer science

This TSM has described how the practical skills of algorithmic thinking and programming can be developed through course content. It is important for the teacher to judge the level of skills their students need to develop to begin the computational solution, and the formative processes during the course that will support students in developing these skills. Additional examples of activities that can develop skills for the computational solution are provided in the "Activities to develop skills for the computational solution" section of the TSM.

Presenting and explaining the IA criteria

The IA criteria are best introduced in class activities throughout the first year of the course. Applying the criteria to smaller assignments, whether individual or collaborative, is another way to familiarize students with the criteria. Students should be given sufficient opportunities to unpack the requirements for each criterion.

Choosing a problem of sufficient complexity

The IA task is an opportunity for students to demonstrate the computational thinking and programming skills they have acquired during the course, and use them for the construction of a computational solution to address a real-world problem. A realistic completion time is about six months.

The selected problem and the proposed computational solution should be complex enough to allow students to demonstrate computer science skills such as problem-solving techniques, combined with the use of data structures, logic conditions, and data manipulation via file processing and databases. The product can use a procedural approach or an object-oriented approach.

The selected problem should be interesting enough to fully engage students in the development process while meeting the assessment criteria requirements. However, the problem scenario should not require technical abilities beyond the students' capabilities. Similarly, students should not introduce unnecessary complexity to showcase specific knowledge or skills if a simpler solution is available.

In some cases, the selected problem may involve changing an existing software solution. If the student chooses to adapt an existing product, there must be a clear rationale as to why the current product is considered inadequate. Then, the modifications required must be substantial enough to require proper decomposition and design. The original code to be improved should be part of the appendix listing the full code; both the original code and the modifications should be clearly highlighted.

Examples of inappropriate project choices

- Creating an unfinished product such as a template, or the first level of a multi-level game
- Database solutions created in a standard database software application using the application's wizard assistant, or without using both complex queries and macros that have been developed by the student
- Spreadsheets without extensive coding in the form of macros developed by the student
- Software solutions that are limited to the basic functionalities of adding, searching, editing, and/or deleting records solely within RAM
- Typical classroom exercises, such as creating a calculator or cash register
- Simple template-based websites
- Designing an overly complex game

Student planning

When planning their IA task, students must use the five criteria as a guide to the stages in the process. For example, there is a specific planning stage that is assessed under a criterion; students therefore need to actively engage in the planning process. They should be supported in understanding the requirements of this and every other stage and criterion, and produce a feasible plan with a chronology and timeline that the teacher can monitor.

This TSM gives specific advice on each of the stages in the section "Unpacking the internal assessment". Each of these stages should be included in the student's plan, allowing them to complete their work in an effective and efficient manner.

IA deadline

The final internal school deadline for students to submit their work needs to account for the time it takes to assess, moderate and submit marks and samples to the IB. Teachers must therefore take the following steps into consideration.

Assessment of the documentation

The solution submitted by each student consists of the documentation, the video and the appendices. The documentation from each student must be thoroughly read and each student's video must be watched.

These are then assessed against the assessment criteria. The documentation, video and appendices must be used by the teacher to authenticate each student's submission.

It is recommended that student work is annotated, or a separate document is created with comments and that a justification is given for the mark awarded for each criterion. This process supports awarding the correct mark for each criterion. Note also that during the process of external moderation, annotations, comments and justifications will help examiners understand the reasoning behind the marks given by the teacher.

Internal standardization and moderation

Internal standardization and moderation aim to ensure that work is marked consistently within a school for a given examination session.

If more than one teacher is teaching computer science in the school, it is essential that internal standardization and moderation takes place.

- **Standardization** refers to two or more teachers marking some representative samples of student work, to ensure that they agree on the standards that should be applied when formally marking all students' work.
- **Moderation** must then take place after marking is complete. Teachers counter-mark some representative samples of each other's marking to check that the agreed standards have been applied. The results of moderation are then applied across all students' work submitted for that examination session.

Submission of marks to the IB

After all students' marks are entered, a randomized set of student work samples is selected for moderation. Teachers work with heads of department and the programme coordinator(s) to ensure that marks are submitted prior to the deadline published in *Diploma Programme Assessment procedures* (updated annually).

Submission of sample work to the IB

The samples of student work chosen by the IB for external moderation must be uploaded before the official deadline, following the procedures indicated in *Diploma Programme Assessment procedures*.

Activities to develop skills for the computational solution

Algorithmic thinking

It is important to develop algorithmic thinking early in the course. Practical activities can be undertaken with students to promote the required skills. These may include the following.

- Presenting a variety of problems to students, ranging from everyday tasks (e.g. cooking, choosing a movie, getting ready for school) to more complex tasks (e.g. mathematics/physics problems, tracking expenses, calculating profit based on investments). Students are guided in identifying key elements such as inputs, possible conditions (logic), processing steps (operations and their correct order) and expected outcomes. This type of practice will guide students in developing a deeper understanding of the problem before attempting to propose an actual solution.
- Encouraging students to propose algorithms for a particular problem (e.g. checking the validity of a date input format via a variety of validation checks) and to use different methods to outline the steps required (e.g. natural language, flow charts, pseudocode).
- Getting students to evaluate and discuss the effectiveness of their proposed algorithms by using different methods such as identifying the parameters of operability for the algorithm or using trace tables.
- Modelling scenarios using (for example) case diagrams, flow charts, bulleted or indented lists and UML diagrams to design for problems in mathematics such as solving quadratic equations, the Newton–Raphson method and the Euler method.

Programming

Short, guided programming practices using programming tools should be incorporated into class time to help students develop the technical skills that will serve as a foundation for constructing a workable computational solution. Some formative mini-projects could include the following.

- Data processing: Creating a database of club members, where each member's record has a very limited set of attributes (e.g. first name, last name, date of birth). These records should initially be stored and manipulated within an array structure, which lends itself to the introduction of sorting and searching algorithms. This can then be extended by discussing the need for (and implementation of) permanent storage in files.
- OOP design: Decomposing a known game such as "four-in-a-row" into its component objects, writing code for class definitions and implementing a main game program that uses these classes to play the game.
- Database design: Constructing an entity relationship diagram (ERD) based on a specific scenario (e.g. a restaurant), followed by a proper database design, including a data dictionary, normalization and the possibility of interrogation or modification using SQL commands.

General advice

Calculating the overall criterion mark

Deciding on an overall mark for a criterion can seem challenging, but it is less so when a clear methodology is used.

When marking the computational solution, read the complete documentation and watch the video first to get a general impression of the work before deciding on the marks to be awarded.

Evidence for a single criterion will usually appear in the section of the documentation dedicated to that criterion, but note that it may also be seen in other sections. Read the documentation more than once, marking the overall task against each criterion. Similarly, watch the student's video at least once. The video can give its own particular impression of the computational solution and provide evidence for awarding marks for criterion D.

A best-fit approach must be used to decide the appropriate mark to award for each criterion. The overall mark awarded for a criterion is **not** an arithmetic mean of the different strands. Rather, it is a holistic judgement reflecting the overall standard of work the student has demonstrated for that criterion.

The level descriptors of the criterion appear as bullet points within the markbands. For example, in criterion A, "The response **outlines** a problem scenario" is in the markband 1–2. Read the level descriptors for each criterion (starting with the lowest level) until you arrive at a descriptor that most appropriately describes the level reached by the student's work.

If a piece of work seems to be between two descriptors, read both descriptors again. Choose the descriptor that more appropriately describes the student's work.

Consider how well the application of command terms has been addressed, as described in the *Computer science guide*.

Once all the level descriptors have been considered as well as the markbands in which they appear, a holistic judgement can then be made about which markband the student's performance falls into for the criterion in question. Once the markband has been identified it can then be decided which mark within the markband is most appropriate.

It is not necessary that all level descriptors within a single markband to be met for a mark within that band to be awarded.

Mark positively—give credit for what the student has done; do not penalize what they could have done or should have done. Record only whole numbers when giving marks; partial marks (fractions and decimals) are not acceptable.

Evidence of programming

During the development of the computational solution, teachers should help students to adopt good programming practices, including the use of meaningful names for variables and functions, consistent indentation and adding comments to important sections of the code (e.g. pre- and post-conditions of procedures and annotations on complex segments of code).

In criterion D (development), students should highlight techniques they have used, including excerpts of their source code together with annotations to explain their algorithmic thinking and a separate justification of its use.

Evidence of programming should include clear screenshots of key sections of the computational solution source code, highlighting specific features or algorithms.

Communication style and documentation length

The computational solution consists of three parts.

- The documentation
- The video
- The appendices

Remember that the documentation and video are used for marking the computational solution, while the appendices is where additional information that is not marked is submitted. Students need to be clear that the appendices must include the full source code for the solution.

The documentation

The documentation needs a title and contents page, and must present a section for each of the five criteria. It is recommended that section titles are simply "Section A", "Section B", and so on, aligning with the lettering of the criteria. Each section should deal with that part of the computational solution and the relevant criterion. It will be helpful if students understand clearly how each criterion relates to the computational thinking process.

The total word count for the documentation should not exceed **2,000 words** and must be stated on the front page. Suggested word limits for each section of the documentation are given in the *Computer science guide*. Diagrams and code with associated comments (embedded in the code or given as annotations) are not included in the word count. However, only necessary diagrams are required for the solution. Including additional diagrams that duplicate work or do not add to the solution's development should be avoided wherever possible.

Excerpts of code and associated comments should be included as evidence of the techniques used to code algorithms and the reasons for their use. These excerpts should be chosen carefully, and a reason should be given for each of them being in the documentation.

The video

It is suggested that students plan the video before recording (e.g. via a storyboard or a simple written plan), to make the most of the five minutes available to them. It is also recommended that students aim to keep the video below 500 MB.

The video must demonstrate the computational solution's functionality and give examples of the testing strategy. The audiovisual quality of the video must be sufficient to demonstrate these; however, students need to be clear that they are not being marked on this quality, but rather on what the video shows. Students should also note that the video is not intended to address the complexity or code of the product, nor possible extensions of the product.

A simple, single test-run of the product will often be sufficient for a student to demonstrate its functionality in the video, followed by some examples of the testing strategy. A straightforward screen recording can often demonstrate these, though it is also recommended that the video comes with commentary, either spoken or as on-screen text. Students are permitted to speed up the video only for specific circumstances (e.g. to show a lengthy sequence of data entry). They should not speed up the video to cram in more information overall, (i.e. by speeding up the rate at which written text appears on screen such that it cannot be read without pausing the video).

Teachers should discuss with students the most effective tools available for recording their computer screens. Different operating systems may have built-in screen recording software, or students can use free screen-recording software available online.

The appendices

The full source code of the computational solution must be included as an appendix. While this code will not be run or marked, it is an important reference for the excerpts of code included in the documentation. For example, the examiner may look at it to clarify ambiguities in the product documentation.

The appendices can also include data and other information referred to in the documentation where students feel this is not sufficiently important to include in the documentation itself—i.e. the part that will be marked. Students need to be clear that no marks are awarded for any content presented in the appendices. For example, the appendices could also include a bibliography if research has been undertaken in computer science (or another domain of knowledge) that has been referenced in the main documentation. However, the presence of this bibliography and its references would not result in higher marks being awarded in the main documentation, though the examiner might find it useful to see these full references.

Criterion A: Problem specification

This section gives advice on the choice of problem for the computational solution and the best ways to address this criterion in the context of the overall solution.

Problem scenarios

The choice of an appropriate problem is key to the success of the computational solution. A suitable problem can come from the outside world, from a different subject in a student's DP studies, or from a computational context. The problem needs to be sufficiently complex for a solution to require use of the computational thinking process and address all the criteria. The problem should also require sufficient innovation such that the solution is demonstrably the student's own work and ideas.

For example, in developing a computer game, over and above the game having sufficient complexity, it should also have a meaningful purpose. This could be an educational game for a specific subject area or a specific target audience. The game should be original and aim to develop a specific skill set. Simply recreating an existing game, or building a new version without significant innovation, will not meet the criteria for the computational solution.

Example problem scenarios may involve:

- inventory management systems
- student databases
- library management systems
- appointment scheduling tools
- encryption/decryption utilities.

The problem can arise from any idea or need related to the students' interests. Once the problem is identified, students must articulate it clearly to construct a problem scenario. Asking some of the following questions may help in this regard.

- What is the current situation?
- What causes the problem?
- Who is affected by the problem?
- What are the objectives or general requirements for the computational solution to this problem?
- Are there any constraints associated with the problem?
- Are there existing computational solutions, and if so, why are they not effective? What are their shortcomings?
- Why is it valuable or significant to construct a computational solution for the problem?
- Are there any time limitations, resource limitations, or specific conditions?

Computational context

Once a problem scenario has been defined the student needs to decide what type of computational response is needed to address it. They will start by choosing a specific programming language or programming environment, depending on the nature of the proposed computational solution. For example, for:

- web-based solutions, use of HTML, CSS, JavaScript, etc.
- database solutions, appropriate programming languages to perform operations on data

- stand-alone solutions and mobile applications, use of C#, Java, Python, etc.

These choices need to be explained in the context of the problem scenario.

Success criteria

The success criteria are a set of measurable software specifications that will determine the effectiveness of the proposed computational solution. They are mentioned in four of the five criteria, so it is vital that students understand what is required in meeting these. The criteria must be derived from the identified problem and its requirements, and should refer to the essential functionality of the software features that will help to solve the problem.

When working on the construction of the success criteria, students must ensure:

- relevance
- clarity
- achievability
- testability.

The criteria should be concrete, easily showcased and demonstrable, align with the testing strategy outlined in the video and specified in the system overview section of the documentation.

Success criteria should not be based on a potential user's perception of the computational solution—the criteria must not be subjective (e.g. how user-friendly the solution should be or what the GUI should look like). If the UI is one of the success criteria this must be clearly stated and measurable with evidence.

Success criteria should be worded in precise terms related to the required functionalities of the computational solution.

Success criteria need to be sufficient in number (around 8–10), substantial, specific and testable.

Examples of good success criteria

- The solution will allow for the input, editing and deletion of customer records from the customer database.
- The solution will give appropriate warning messages to the user in case of extreme or invalid inputs.
- The solution will be menu-driven with eight menu choices (including an indication of the choices).

Examples of inadequate success criteria

- The code will compile.
- The code will run without crashing.
- The code will use loops.
- The program will be aesthetically pleasing.

Criterion B: Planning

Decomposing problems

This part of the computational thinking process is important in breaking down a complex problem into a number of more manageable problems. The decomposition will also inform planning, and any plan will involve the order and requirements of each of the decomposed problems.

Decomposition involves identifying the main components of the solution as well as its functional parts. Once the main components have been clearly identified, students should divide each component into smaller and more specific tasks. This approach reduces complexity, making each task easier to tackle.

Decomposing problems into smaller, manageable sections is a fundamental step in the computational thinking process. By doing this, students can better plan and execute the development of their computational solution.

There are multiple tools to facilitate decomposition.

- The general solution can be documented by a use case diagram together with flow charts, data flow diagrams and/or the equivalent.
- For procedural coding it may be helpful to use a flow-charting tool to break down a complex task into modules.
- For object-oriented coding it is important to identify and describe the objects in the problem scenario.
- A database project requires identifying and describing the data, and the data processing needed.

Planning the computational thinking process

Once a problem has been decomposed it is possible to plan for its solution. The plan should outline the order of tasks, the timeline for their completion and the resources needed to complete individual tasks. Any research that is required can also be planned for.

The plan should clearly address the steps involved in all five stages of:

- planning
- designing
- developing
- testing
- evaluating.

A GANTT chart can be used to map out the steps in the plan. It is also important to maintain an AGILE approach where possible, so that the documentation matches the reality of the proposed solution and addresses the details. A generic GANTT chart will not fulfil this criterion.

After decomposing a problem, students should create a detailed chronological plan, considering:

- task order and dependencies, to ensure efficient task execution
- success criteria to address all requirements for a successful solution
- subtasks that break down major activities into steps for planning, designing, developing, testing and evaluating
- time, limitations and resources to allocate realistic timeframes, consider constraints and identify resource needs
- planning of research to integrate any necessary research activities within the overall plan.

Students need to be clear that the plan is not a process journal. Also, while an AGILE approach is encouraged, the planning stage of the documentation should not be written after the computational solution is completed, but beforehand—it is a plan from the outset that guides development of the solution.

Criterion C: System overview

System models

UML diagrams of system models depend on the computational context of the solution. It is important that students choose the appropriate diagram style for their chosen computational context. They should also be encouraged to use the appropriate number of diagrams, as additional unnecessary diagrams can detract from the solution.

The system model, along with its individual components in the system overview, should be sufficiently clear that a third party with the necessary programming knowledge would be able to complete the solution. The system model on its own is a higher-level view of the system showing how the individual components connect with each other. The model will be related to the decomposition of the problem, as each component will represent a solution of the sub-problems in the decomposed problem. There are multiple tools to facilitate this.

- For procedural coding, a flow-charting tool can break down the modules from the planning stage.
- For procedural coding, a bulleted list with indentations might be used.
- For object-oriented coding, it is important to document the component objects using UML diagrams that show dependencies between objects.
- A database project would require table design with proper normalization and a description of data dictionaries.

Annotated design sketches of the UI should be part of this stage of the design process.

Algorithms as components in a system model

The individual components in a system model are made up of algorithms. Each of these algorithms should be clearly seen individually in section C of the computational solution and relate directly to the components in the system model.

The design of specific algorithms can be shown as:

- pseudocode
- flow charts
- bulleted lists with indentations.

Testing strategies

At this point in the planning stage, students need to consider testing strategies. A variety of rigorous tests need to be developed to assess the functionality of the product against the success criteria as stated in the problem specification (criterion A).

In addition to functional testing, the product needs to be tested structurally using a variety of valid, extreme and invalid data that covers all possible situations.

A testing strategy for functional testing could be fulfilled with a table that lists the success criteria, and a separate column with descriptions of how each separate success criterion will be tested.

A testing strategy for structural testing should focus on the major algorithms and whether they perform correctly. This could be fulfilled by adding tables that specify valid, extreme and invalid input data, together with a column for the expected results.

Comprehensive structural testing also requires students to address all eventualities. For example, when adding to a sorted list, data needs to be added to the empty list, to the end of the list, to the start of the list and in the middle of the list. The following data would structurally test the add method for a sorted list of animals: horse, zebra, donkey, mule.

Criterion D: Development

Functionality

Functionality refers to how well the product works. The product should address the problem described in criterion A. To show functionality, the video should preferably include a single test-run that shows the performance of the solution following the testing strategies developed in the system overview.

A product that does not address the problem specification (criterion A) cannot achieve more than limited functionality. A product that addresses an inadequate or vague problem specification will only achieve partial functionality. Full functionality can be achieved if the product is a working solution to an adequate problem specification, even if the success criteria have not been fully met.

Functionality will be assessed on the quality of the test-runs in the video and on the evidence of testing included for criterion D.

Algorithms and techniques

The process of turning algorithms into code is a key element in developing a product that solves the stated problem. This process will require students to use different programming techniques. It is essential that they clearly demonstrate these techniques in writing and by showing excerpts of code. Students should provide comments with the code to explain its purpose.

This criterion gives students the opportunity to show their knowledge and understanding of tools and techniques, which must be explained sufficiently to fulfil the intention of "development". There must be explicit evidence of algorithmic thinking in the form of code excerpts, along with explanations and justifications of the chosen techniques.

It is also important that students discuss and evaluate the choices they made when deciding which techniques were most appropriate.

It is acceptable to improve code with a large language model (LLM) such as ChatGPT, but any code, content or ideas generated by the LLM must be acknowledged. Students must include a comment in their code and a note in their documentation indicating where and how they used the LLM.

Student example 1:

"I wrote a function to sort a list, but it had a logic error. ChatGPT suggested using Python's sorted() function, improving efficiency from $O(n^2)$ to O(n log n). I've acknowledged this change in my comments."

Student example 2:

"I needed to write a function to calculate the Fibonacci sequence, but I wasn't sure how to implement it. I asked ChatGPT to generate the code, and it provided a recursive solution. I used this code in my project and added comments to indicate that ChatGPT generated it and to explain how the algorithm works."

Note that documenting the development should **not** be a process journal. Instead, the student must highlight, explain and justify specific tools and techniques used.

Effective testing strategies

The effectiveness of the testing strategies used is documented and discussed in this section. Evidence of the implementation of at least three separate test cases must be included, showing both functional and structural testing.

This could be addressed by copying the test tables for a few major algorithms from the testing strategies section in the system overview, adding a new column for the observed results. These results should address valid, extreme and invalid data. It is expected that students will refer to evidence of testing seen in the test-runs in the video.

Students can justify the effectiveness of testing strategies by evaluating the comprehensiveness of the testing.

Criterion E: Evaluation

Meeting success criteria

Students may want to include a table that lists all the success criteria together with an in-depth evaluation addressing to what extent each criterion has been achieved by the solution. The evaluation should refer to evidence of testing as seen in the video and/or in the documentation.

Suggesting improvements

Suggestions for improvement will address issues with the final solution that came to light during testing. Students should justify these suggestions by referring to how the improvement will address the specific issues. The suggestions for improvement can be regarded as appropriate and adequate when they are specific and actionable in addressing the issue that was identified.

Examples of inadequate suggestions for improvements would be to:

- merely include more colour, more data, more calculations, more functionality (unless very specifically justified)
- add functionality for success criteria that have not been met
- add a GUI for a solution that was developed as a CLI
- add functionality that was always an essential component of a fully functional solution but that was previously missing (e.g. a payment function for a web shop, or permanent storage for a database).

Bibliography

Banchi, H., & Bell, R. (2008). The many levels of inquiry. *Science and Children*, *46*(2), 26–29.

Boud, D., & Feletti, G. (1997). *The challenge of problem-based learning* (2nd ed.). Routledge.

International Baccalaureate. (2020). *IB learner profile.* International Baccalaureate Organization. https://resources.ibo.org/permalink/11162-43492?lang=en

Kolb, D. (1984). *Experiential learning: Experience as the source of learning and development*. Prentice Hall.

Project Zero. (2015). *Project Zero's thinking routine toolbox*. Harvard Graduate School of Education. https://pz.harvard.edu/thinking-routines