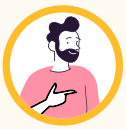
Kognity

IBDP Computer Science

Teacher to Teacher Tips

Kognity is designed to help you prepare students for success while saving planning and feedback time. The questions below reflect common concerns from IBDP Computer Science classrooms, with practical suggestions for using the book, assignments, Practice Centre and Insights.



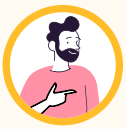
Should I introduce computational thinking before asking students to code?

While learning any IB CS content, start with the real-world scenario and problem specifications in each subtopic. Students first define the problem, objectives, constraints, inputs, outputs and measurable evaluation criteria. They can then apply decomposition, pattern recognition, abstraction and algorithmic thinking. This keeps the focus on the quality of the solution before syntax becomes a distraction.

- Use computational thinking during programming sections to create a solution to a problem.

How should a problem statement be worded?

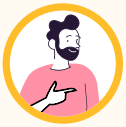
Section	Good examples	Bad examples
1. Problem statement	✓ Students cannot easily find available study rooms on campus. ✓ Paper invoices cause delays and errors in payment processing.	✗ The system is not good. ✗ We need a better way to do things.
2. Objectives and goals	✓ Allow students to search and book study rooms online. ✓ Reduce invoice processing time by 50%.	✗ Make the system better. ✗ Improve performance.
3. Constraints and limitations	✓ Must run on devices with at least Android 6.0. ✓ Budget is limited to $10,000.	✗ There are some technical limits. ✗ We don't have a lot of money.
4. Input specifications	✓ Student ID (text), entered by user. ✓ Room number (integer), selected from a list.	✗ User enters information. ✗ All necessary data.
5. Output specifications	✓ Booking confirmation displayed on screen. ✓ Email confirmation sent to the student.	✗ Show results to the user. ✗ Print reports when needed.
6. Evaluation criteria	✓ A room can be booked in under 30 seconds. ✓ System uptime must be at least 99%.	✗ The system should be fast. ✗ Users should be happy.



How can I support a class with very different levels of programming experience?

Keep the learning objective common, but vary the level of support and extension. Beginners can annotate worked examples, trace code and use the glossary or Immersive Reader. More experienced students can complete extension tasks, compare alternative implementations or explain how the code could be improved. Pair discussion can help, provided every student produces an individual explanation afterwards.

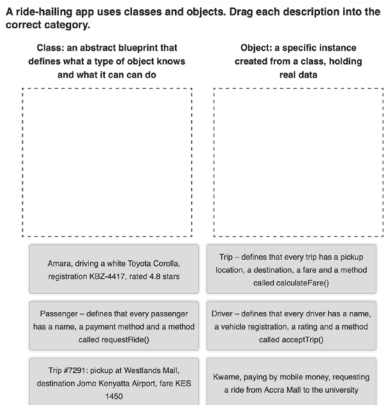
- Assign the same concept with different starting scaffolds.
- Use extension suggestions in the included programming projects to differentiate.

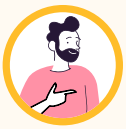


How can I teach Java or Python to students with no prior experience?

During complex content, teach the language-independent concept first. Use pseudocode, diagrams or a verbal model, then compare the Java or Python examples side by side. Ask students to highlight the lines that perform the same role, such as declaring a child class, calling a parent constructor or overriding a method. A comparison task can therefore reinforce the programming languages too overwhelming.

- Focus on programming fundamentals before introducing more complex concepts.
- Spend more time on learning objectives that teach the basics, before moving to other content.

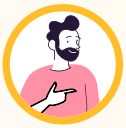




How can I use Kognity to create effective formative assessments?

Issue question assignments for a particular subtopic, or use section questions as an exit ticket. Students receive immediate feedback, while you can use assignment and Insights data to identify common misconceptions. Begin the next lesson with one anonymised error pattern and ask the class to correct it together.

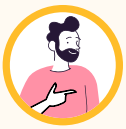
- Use a short pre-assessment before a new topic.
- Use the same concept again after teaching to measure improvement.
- Create targeted follow-up work for students who need additional practice.



How should I use exam-style questions?

Use them formatively or summatively. You can select specific questions, set instructions and time limits, and control when the markscheme is released. For formative practice, release the markscheme after submission and ask students to annotate where marks were gained or lost. For a summative task, keep the markscheme hidden and mark the written responses using the provided guidance.

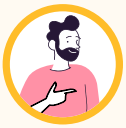
- Model timed conditions gradually rather than beginning with a full paper.
- Ask students to identify the command term and plan the structure of the response before writing.



How can I encourage students to take ownership of their learning?

Make progress information part of the classroom routine. Ask students to review their strength bar, assignment feedback and notebook once a week, then choose one specific subtopic or skill to improve. The target should name an action, such as rereading a section, completing five questions or explaining a concept without notes.

- Use a short reflection after each unit.
- Have students keep a personal list of recurring coding or response errors.
- Celebrate evidence of improvement, not just high scores.



How can I prepare students for the Internal Assessment through normal lessons?

Build IA habits into regular classwork. Each subtopic provides tool tips to practice in defining a problem and writing measurable criteria. Programming activities develop decomposition, design, implementation and testing. Discussion and reflection tasks help students justify choices and evaluate limitations. Treat these as recurring skills rather than waiting until the IA begins.

- Ask students to explain the user need behind every proposed feature.
- Require testable success criteria for small projects.
- Have students record design decisions and revisions in the notebook.