Raspberry Pi Foundation

Pedagogy Quick Read

Computational Thinking 2.0

Computational Thinking (CT) has become a cornerstone of computing education. CT2.0 has recently been introduced by Matti Tedre and his team[1] to help learners distinguish between traditional rule-based approaches (CT1.0) to problem solving, and the data-driven approaches (CT2.0) used by AI systems. As systems increasingly include both rule-based and data-driven elements, it is essential for learners to understand the differences and to be able to work with both paradigms.

Computational Thinking 1.0		**Computational Thinking 2.0**
Correctness can be formally proven	Correctness	Models may display higher or lower confidence
Debugging: tracking and tracing	Debugging	Evaluate the model with regards to predictions
Deductive problem-solving	Problem solving philosophy	Inductive problem-solving
Transparent structure	Structure	Black box
Stepwise, deterministic, discrete flow of program through states	Notional machines	Parallel, network-based data processing with likely non-deterministic
Structured	Data	Unstructured

Image based upon original comparison from Tedre, Denning, and Toivonen[1]

Problem solving

CT is a framework for understanding problem solving using computation. Traditional CT1.0 emerged from early computing concepts, using a rule-based approach where computer programs follow precise instructions: with a well-defined input, instructions are followed step by step to produce a predictable output. When teaching with CT1.0, learners learn to break down the task into subtasks and write clear instructions for each step before implementing these instructions in tools like Scratch and languages like Python.

In contrast, problem solving in CT2.0 shifts to a data-driven approach[1]. Rather than writing explicit instructions, learners learn to collect, clean, label, and organise large amounts of relevant data. Learners then use this data to train machine learning (ML) systems to identify patterns and produce models that generate predictions and solve problems. For example, in CT1.0, learners could create a tool that classifies cats using If...Then rules about whiskers and pointy ears. However, in CT2.0, they would use many images of cats to train a model with sufficient accuracy. Because data is central to this process, data quality, evaluation, and bias become critical concepts: flawed or biased datasets can lead to unreliable or unfair outcomes.

Modern applications combine both rule-based and data-driven approaches — from AI-generated text and images, to face recognition software and social media recommendations. Understanding both CT1.0 and CT2.0 empowers learners not only to work effectively with these tools, but also to be active participants and creators rather than passive consumers in our increasingly data-driven societies[2].

Key concepts

Problem solving

- CT1.0 applies rules-based approaches to problem solving, like those used in Scratch and Python.
- CT2.0 presents a shift towards a data-driven approach to problem solving.
- Model evaluation, data quality, and bias become important in CT2.0, as flawed data can lead to unfair outcomes.

Correctness

- CT1.0 typically teaches correctness as binary, where programs do or don't produce correct outcomes.
- CT2.0 measures correctness by degree, where ML models generate predictions and levels of confidence.

Debugging

- In CT1.0, debugging is structured and transparent. Errors are addressed by tracking the program execution step by step.
- In CT2.0, ML models are opaque black boxes. Problems are found through analyses of the input and output data.
- This requires a shift in the debugging mindset, focusing on improving training data, tuning parameters, and testing with a diverse range of data.

Understanding correctness

Correctness is an important concept in computing and determines whether a program functions as intended. In CT1.0, we often teach learners that correctness involves a program being either correct or incorrect. This approach emphasises precision, where instructions must be syntactically correct, written logically, and produce the expected outcome. Rule-based programs characterised by CT1.0 assume a high level of transparency: every instruction is explicitly written and can be traced back, errors can be pinpointed, and corrections can be tested and implemented.

In CT2.0, correctness is no longer a fixed correct-or-incorrect. Outcomes in many ML models are probability-based predictions with varying levels of confidence[1]. For example, an ML model might classify a picture of a cat with a 95% confidence score. Even well-trained ML models, despite being trained on large amounts of data, could produce errors, especially with new inputs. For example, an image of a cat could be incorrectly classified as a dog with a 60% confidence score. Developers define acceptable levels of correctness when designing and building ML models. This requires developers to carefully tune the training process and set appropriate confidence thresholds to determine whether a prediction is acceptable for a specific context.

For educators, this shift in understanding correctness requires helping young people to develop critical thinking skills around data-driven tools and AI systems. We could guide learners to ask deeper questions: "How reliable is this prediction with new data?" or "What biases might be in the training data?" By framing correctness or suitability in CT2.0 as an ongoing process of evaluation and continuously refining models to improve their reliability in real-world applications rather than a fixed outcome, we prepare learners not just to use AI tools, but to recognise system limitations and potential harms caused by system outputs.

Debugging

Debugging is another practice that takes on different forms in CT1.0 and CT2.0. For example, if a rule-based program implemented in either Scratch or Python doesn't work as expected, learners can display variable values, set breakpoints, or trace the code line by line to find where things went wrong. Because of the high level of transparency in such programs, we can use systematic and structured debugging practices.

However, ML models are often seen as black boxes[3], and this opacity makes debugging in CT2.0 less straightforward. ML models are complicated, interconnected networks with billions of parameters that determine outcomes and predictions in ways that are impossible to trace step by step. When an image classifier incorrectly labels an image of a cat as a dog, learners can't simply find the line of code causing the error because there isn't one. Instead, debugging in CT2.0 involves examining and improving the quality of the training data, tuning variables and parameters, and testing with a range of diverse inputs to identify patterns in errors (e.g. cats with pointy ears are more likely to be misclassified as dogs). Debugging now requires educators to shift from finding bugs and error correction to focusing on how changes to data and parameters can affect overall performance.

Why CT2.0 matters

Without CT2.0, today's learners will remain passive consumers rather than informed participants in a world increasingly shaped by data-driven AI technologies. Integrating CT2.0 alongside traditional computational thinking provides learners with an accurate understanding of computing systems, including how problem solving, correctness, and debugging differ in data-driven systems.

This will empower learners to critically evaluate ML models, understand how data is used to train models, identify potential biases, and even create their own ML projects. Embracing CT2.0 makes computing more realistic and representative of the real world, offering learners pathways beyond traditional programming and towards future careers where AI literacy is crucial.

References

1. Tedre, M., et al. (2021). CT 2.0. the-cc.io/qr21_1
2. Vartiainen, H., et al. (2021). Machine learning for middle schoolers: Learning through data-driven design. the-cc.io/qr21_2
3. Hitron, T., et al. (2019). Can children understand machine learning concepts? The effect of uncovering black boxes. the-cc.io/qr21_3

Raspberry Pi Foundation