Computational Thinking Framework 2018

let's talk science | parlons sciences

Acknowledgments

This framework was created based on research, resources and ideas from around the world, and is considered current as of October, 2018. As the thinking relating to teaching and learning Computational Thinking evolves, this framework will be revisited to ensure that it continues to provide meaningful information to educators.

Let's Talk Science would like to acknowledge Lisa Floyd, Director of Research and Inquiry and Derek Tangredi, Director of Integrated STEAM Education of Fair Chance Learning for sharing their extensive knowledge and experience in Computational Thinking. Their contributions are greatly appreciated. Let's Talk Science would also like to thank Kim Taylor, Education Specialist, for the development of this framework as well as all of the staff who contributed their expertise to the review of this document. For more information about Let's Talk Science, please visit:
letstalkscience.ca.

Let's Talk Science
1510 Woodcock Street, Unit 12
London, Ontario, Canada
N6H 5S1
letstalkscience.ca

1.1 Computational Thinking: An Introduction

Virtually every sphere of our work and life has come to be dominated by digital technology. Young people have grown up never knowing a world without computers and smart phones. Even though many youth are avid consumers of technology, there is a growing concern that they are not being prepared to become producers of technology. Kafai & Margolis (Washington Post online, 2014) described it this way:

"Being a digital native today isn't just about browsing the web, using technology to communicate, or participating in gaming networks. It really involves knowing how things are made, breaking down and solving problems, designing systems, contributing through making, and understanding social and ethical ramifications. We see how computers in any form and place have become an inextricable part of our social lives—not just how we interact but also how we contribute."

As technology advances, it will become even more important that young people are prepared to contribute to their digital world. This includes, and is not limited to, being prepared to interact in a meaningful way with others, taking part in a workforce in which computers play an ever-increasing role, making decisions about how technology shapes their world, and finding solutions to problems that face us all.

Educators will need support to develop an understanding of what is important for our youth to learn so that they may play an active role in the ever-changing digital world. We would argue that in the 21st century and beyond, students will need to learn to think computationally, that is, to develop the skills, knowledge and habits of mind of Computational Thinking (CT).

The term "Computational Thinking" is beginning to appear in education systems around the world, including the Canadian K-12 education system. As CT will be new for many educators, it will be important for them to understand what CT encompasses so that they will be prepared to effectively support student learning. A number of resources from around the world were examined for their definitions and explanations of CT. These included curriculum documents, academic literature, resources for educators, and websites of CT organizations offering CT-based programs.

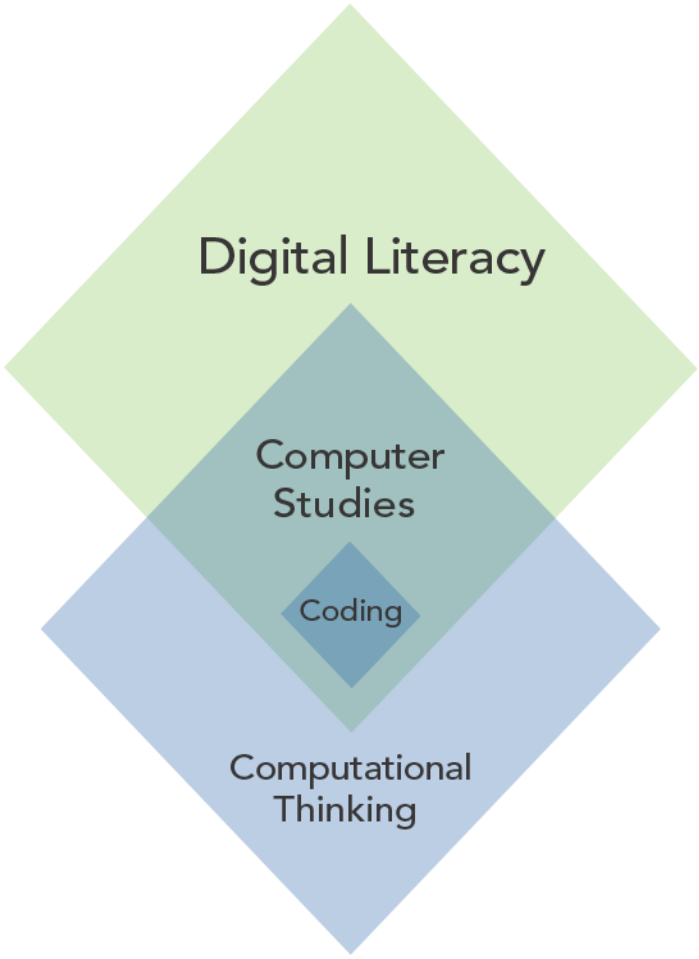
A rich CT environment is one in which a user can develop skills to become a creator. (Lee et al 2011, p.35)

For information, definitions and sources cited in this document see the Let's Talk Science Computational Thinking Literature and Curriculum Review (Let's Talk Science, 2018).

1.2 Where Does Computational Thinking Fit?

In most respects, CT falls within Digital Literacy (DL). According to the Information and Communications Technology Council (ICTC), DL is "the ability to locate, organize, understand, evaluate, and create information using digital technology for a knowledge-based society" (2012, p. 4). The Brookfield Institute for Innovation + Entrepreneurship defines DL as the "ability to use technological tools to solve problems, underpinned by the ability to critically understand digital content and tools. This can include the more advanced ability to create new technological tools, products and services" (2017, p. 11). It is important to note that some aspects of CT can be developed without digital technology, which is why the DL and CT squares in Figure 1 do not overlap completely.

Computer Studies (CS) is less "about learning how to use a computer" and "much more than computer programming" (Ontario Ministry of Education, 2008, p.3). Computer Studies, which the Ontario Ministry of Education considers to be "about how computers compute," occurs where DL and CT overlap, as depicted in Figure 1. Computer programming, known more commonly by its more user-friendly term "coding," falls squarely within DL, CS and CT. Coding is about telling computers what to do. In most cases, people use a specific programming (or coding) language which defines how the code should be written so that the computer can understand it. Coding can help students develop many aspects of CT.



Digital Literacy includes computer studies and coding, and overlaps Computational Thinking.

1.3 What is Computational Thinking?

There is currently no universally accepted definition of CT. There are many similar yet different definitions and explanations of CT which reflect the interests and opinions of researchers, educators and organizations (computer science, science, mathematics, etc.).

In many cases, instead of defining CT outright, sources explain CT in terms of what students must know and do to develop CT. Sometimes the learning outcomes are conceptual (e.g., concept of algorithms) and sometimes the learning outcomes are skills-based (e.g., algorithmic thinking). Sources also describe CT-based learning activities which they label strategies, methods, approaches, processes, and practices.

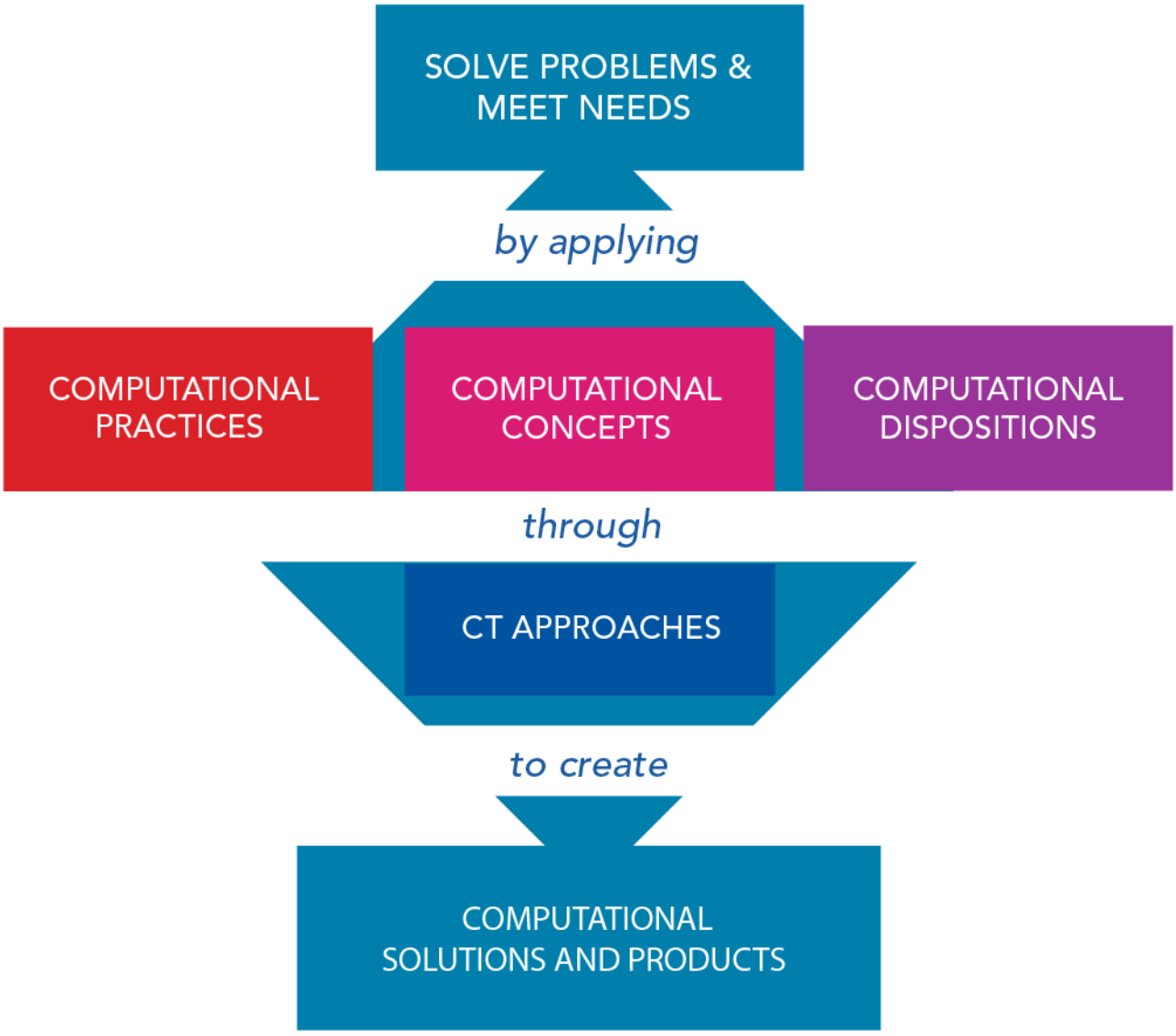
Although there seems to be little consensus on terminology, there are some recurring ideas in the research.

Computational Thinking...

involves thinking processes such as logical reasoning	is associated with, but not limited to, problem-solving; including defining, understanding, and solving problems	draws on the tools, techniques and concepts of computing (e.g., decomposition, abstraction and algorithmic thinking)
involves systematically and logically structuring procedures (algorithms) to generate automatable (programmable) solutions	enables the creation of solutions that can be effectively carried out by an information-processing agent, such as a computer	often involves gathering, organizing (sorting, grouping) and analyzing data (looking for patterns, dependencies and relationships)
may lead to the creation of digital products, processes and systems	develops and supports higher-order thinking skills such as analysis, synthesis and evaluation	supports development of 21st century/global skills and competencies including creative thinking, critical thinking, collaboration and communication

1.4 Let's Talk Science Computational Thinking Framework

The Let's Talk Science CT Framework is a synthesis of the many definitions and explanations that have been proposed to date. It is influenced by work done by the Computer Science Teachers Association (USA), the International Society for Technology in Education, Computing at School (UK), the Brookfield Institute (Canada), Karen Brennan and Mitchell Resnick (MIT media lab), and Peter J. Denning. It is designed to illustrate how the various facets of CT enable students to develop the skills, understandings and habits of mind they need to solve problems and meet needs in the digital world.



The following lists in more detail what one would expect to see in each of these areas as students develop Computational Thinking.

Computational Practices (Skills)

- Decomposition
- Abstraction
- Pattern Recognition
- Algorithmic Thinking
- Testing & Evaluating
- Logical Thinking (involves reasoning)
- Debugging
- Data collection & Analysis
- Data Representation

Definitions are found in Section 1.5

Computational Concepts (Understandings)

- Sequences
- Repetition (loops)
- Conditionals
- Data
- Variables
- Events
- Operators
- Functions
- Inputs & Outputs

Definitions are found in Section 1.6

Computational Dispositions (Habits of Mind)

- Persistence
- Comfort working with others
- Comfort with trial & error
- Flexibility
- Creativity
- Ability to tolerate ambiguity
- Ability to deal with open-ended problems
- Confident dealing with complexity
- Inquisitiveness/curiosity

Computational Thinking Approaches

- Unplugged (without computing devices)
- Tinkering
- Reusing & Remixing
- Making

Definitions are found in Section 1.7

Examples of Computational Solutions and Products

- Digitally-controlled physical objects
- Websites and mobile applications
- Computational models and simulations
- Computer algorithms and programs
- Data models, visualizations and structures

1.5 Definitions: Computational Practices

Decomposition

Definition:

Decomposition involves breaking down a problem into smaller parts or sub-problems. Working with smaller subsets of a problem can reduce the overall complexity of a problem. Decomposition may also involve thinking about computational products in terms of their component parts (e.g., graphics, data, user interface). Not only does CT involve decomposing a problem, it also involves applying knowledge of how previous problems were solved and composing a solution to the problem after all of the sub-problems have been tackled.

Examples:

- *Identify all of the tasks you will need to do in order to bake a cake.*
- *Explain how the parts of a dragonfly make it an effective predator.*
- *Create a video game with your friends. How could you divide up the tasks?*
- *Create a code that will convert a value in Fahrenheit to one in Celsius. What are the major parts and processes?*

Associated with:

- Deconstructing, Dividing, Sorting, Classifying

Abstraction

Definition:

Abstraction is about reducing the complexity of a problem or task by focusing on what is important, capturing relevant information and removing unnecessary details. Abstraction can also be used to have one object stand for many, or to have a word stand for an action. Models and simulations can also be considered abstractions.

Examples:

- *Identify the most important skills for a baseball pitcher.*
- *Represent the actions run, stop and hop using images.*
- *Code a simulation of a volcano erupting.*
- *Identify the appropriate formula to use to solve a physics problem.*

Associated with:

- Deconstructing, Dividing, Sorting, Classifying

Pattern Recognition

Definition:

Pattern recognition involves being able to recognize and use patterns to describe and represent sequences in data or processes. By identifying patterns, predictions can be made as to how things might work or what might happen in a given circumstance. Identifying patterns enables the creation of rules, such as when actions can be repeated automatically. Pattern recognition also enables previously successful methods to be applied to new problems, which can improve the efficiency of the problem-solving process.

Examples:

- *What pattern do you notice when you draw a square? How could you apply that knowledge to drawing a pentagon?*
- *Which notes are repeated in the music? When are those parts repeated?*
- *In baseball, what happens after you receive two strikes? What about three strikes?*

Associated with:

- Observing, Predicting, Comparing, Generalizing

Algorithmic Thinking

Definition:

Algorithmic thinking is the skill involved with creating an algorithm. An algorithm is a series of ordered, logical and unambiguous rules or instructions necessary to solve a problem or achieve an objective. By identifying steps that can be communicated as instructions (verbal or written), codes or programs to other people or to computing devices, algorithmic thinking skills are employed.

Examples:

- *What steps are needed to make a sandwich? Does it matter if the steps are done in a different order? Why or why not?*
- *Create an algorithm that would help you decide what to wear each day.*
- *Create an algorithm that can help you identify the birds in your backyard.*
- *Create an algorithm to teach a young child how to tie shoelaces in a bow.*

Associated with:

- Planning, Organizing, Sequencing, Classifying, Sorting

Logical Thinking

Definition:

Logical thinking means thinking using logic. It involves starting with a premise (information that you believe to be true) and then using reasoning to come up with a conclusion that makes sense based on the premise. We apply logical thinking when we use abstraction, algorithmic thinking and pattern recognition.

Examples:

- ***Deductive:*** *uses a specific premise that leads to a specific and accurate conclusion (e.g., All squares are rectangles. All rectangles have four sides; therefore, all squares have four sides.).*
- ***Inductive:*** *uses a specific premise that leads to a broad and not necessarily accurate conclusion (e.g., The grade 6 students in the class like gym period; therefore, all grade 6 students like gym period.).*

Associated with:

- Classifying, Analyzing, Generalizing

Testing and Evaluating

Definition:

Testing involves trying something and observing what happens. **Evaluation** is about using critical thinking and judgement to determine if a set of criteria are met. If not, then correction or improvement may be required. Computer programming tends to be an interactive process in which testing and evaluating is constantly occurring.

Examples:

- *Did the animated character move in the way you expected it to?*
- *What happened when you tested the robot? Was it able to shoot the ball into the net every time?*
- *What happened when you downloaded the code for scrolling your name onto the micro:bit? Did it scroll the letters of your name across the LED display in the correct order?*

Associated with:

- Observing, Comparing, Analyzing

Debugging

Definition:

When a computer code does not do what you think it is going to do, it may contain errors. Errors in computer code, or programs, are known as 'bugs.' **Debugging** involves finding and correcting bugs. Bugs include syntax errors (e.g., spelling mistakes), logic errors (e.g., incorrect logic) and other types of errors. Debugging requires logical thinking.

Examples:

- *Why does the computer say that the code doesn't compile, is it because of a spelling mistake?*
- *Why does your program give an unexpectedly large answer, is it because of a logic error?*
- *What simple tests could you run to help you find the mistake in this program?*

Associated with:

- Observing, Testing, Analyzing, Interpreting

Data Collection and Analysis

Definition:

Data collection is a planned approach of selecting and gathering information in an effort to answer a question or solve a problem. When planning for documenting and organizing data, it is important to consider the types of data to be collected (e.g., qualitative, quantitative) as well as how best to document and organize the data so that it will be useful for analysis and interpretation.

Analyzing data involves observing data to determine patterns such as repeated results, upward and downward trends, etc. Analysis also includes trying to explain patterns and trends as well as unusual points or discrepancies in the data.

Examples:

- *How many animals cross the road in this location after dark? How will you collect this data?*
- *What do you notice about the germination of our seeds over the last two weeks?*
- *What is the most common species of bird at our class bird feeder? What data would you need to collect? How will you collect it?*
- *What was the average time it took for the toy cars to go down the ramp?*

Associated with:

- Observing, Comparing, Analyzing, Interpreting, Concluding

Data Representation

Definition:

Once data has been collected, it is often stored in a computer database. At times users may want insight about all of the data in a database or only about a certain subset of the data. In order to be able to analyze and communicate findings, it is often useful to **represent the data** in a more human-friendly format such as a table, chart, graph, infographic, etc.

Examples:

- *What cities in Canada have the highest level of carbon dioxide in their classrooms? What is the best way to represent this data?*
- *How would you organize data about daily rainfall in major cities to help researchers compare rainfall between time periods and places in the future?*
- *What is the most useful way to organize your math test scores to help you analyze your progress?*

Associated with:

- Organizing, Sorting, Interpreting

1.6 Definitions: Computational Concepts

Sequences

Definition:

The steps in an algorithm always follow a **sequence**. It should be noted that sequences are rarely linear and may involve tasks which are repeated or only occur under certain conditions. A flowchart is a good tool for understanding and representing sequences.

Simple, linear sequencing is one of the first concepts learned when coding.

Examples:

- *What are the steps you need to do to……? (e.g., change a tire, brush your teeth, make toast). Are there any steps that you repeat?*
- *Writing out the rules for a game (e.g., what happens first, next, last)*
- *Describing how you get to school using a map (e.g., where do you start? When do you turn? Where do you end?)*
- *Explain to someone else how to draw a square of a specific perimeter.*

Repetition (Loops)

Definition:

Repetition involves repeating a step or steps in an algorithm a certain number of times until a certain predetermined end point is reached. Repetitive tasks are very common in computer programing and setting up tasks so that they automatically repeat (loop) can save a lot of time. Loops help programs be more organized and shorter because they cut down on the amount of code that needs to be written.

Examples:

- *Explain to someone else the most efficient way of making 100 sandwiches.*
- *Set up a light to turn on and off at certain times every day.*
- *Code a car game so that the car will automatically go around the track 10 times.*
- *Create a piece of art that involves a repeating pattern, like a mosaic tile, mandala, or fractal.*

Conditionals

Definition:

At times in algorithms, there may be a need to select one action out of a set of actions, such as which way to go at a fork in the road. Conditional statements give rules to direct the flow of what happens, such as **if** something is true, **then** something will happen or **else** something else will happen. Conditionals allow a program to make decisions and direct the flow of activities without human intervention.

Examples:

- *If it is raining outside then you will need to put on your rain boots, else wear your running shoes.*
- *If the sensor reads a temperature greater than 25°C, then show a happy face display, else if the sensor reads a temperature less than 25°C then show a sad face display.*
- *If you get to a fork in the maze then turn left else go straight on.*
- *If (hour < 18:00) {greeting = "Good day";} else {greeting = "Good evening":}*

Data

Definition:

Computers and other communication systems work with different types of data. **Data** are values that computers can store and retrieve. **Data types** are computer language-specific, but they commonly include types such as characters (letters, punctuation, spaces), strings (sequences of characters), integers (whole numbers), fixed and floating point numbers and the Boolean data type, which is often used with conditionals.

Examples:

- ***Character*** *(e.g., A);* ***String*** *(e.g., Hello!);* ***Integer*** *(e.g., 12)*
- ***Floating-point number*** *(float) (e.g., 15.2203829) - can have a variable number of digits after the decimal*
- ***Boolean*** *(true or false value) - can be understood to mean on/off or start/stop*
- *If you want to keep track of how many goals were scored, which data type will you use? (integer)*
- *If you want to store information about whether the light is on or off, which data type will you use? (Boolean)*

Variables

Definition:

Variables are places where you can store and retrieve data. Variables have names that stand in for the data they hold, which makes them an abstract concept since one thing (variable name) represents something else (data). Variables also have a data type (the kind of data that can be stored) and a value (refers to what is stored in the variable). In coding languages, variables are written as case-sensitive single words with no spaces.

Examples:

- *var a = 10; var b = 5; a and b are the names of the variables and the = tells you what data is stored in each variable.*
- *The variable "age" could represent people's ages (e.g., var age;).*
- *The variable "rateGrowth" could be used to keep track of a plant's growth over a certain period of time (e.g., var rateGrowth;).*
- *The variable "temperature" could be used to store and retrieve the temperature readings from a sensor.*
- *The variable "StudentName " could be used to store a student's name and insert it automatically in a personalized email.*

Events

Definition:

An **event** involves having one action cause another action, such as how you get a computer to respond to the input of a user. User inputs include actions such as clicking a mouse, tapping a key or touching a screen.

Examples:

- *Zooming in on a map when someone clicks on it*
- *Scrolling through images on a phone when someone swipes the screen*
- *Adjusting the volume on a video when someone presses the + or - buttons*

Operators

Definition:

Operators are characters that tell the computer to perform certain mathematical or logic-based actions. The specific operators depend on the programming language but may include operators for adding (+), subtracting (-), multiplying (*), dividing (/), less than (<), greater than (>) and equal to (==), as well as logic-based operations such as AND (&&), OR (II) and NOT (!).

Examples:

- *What is the sum of the ages of everyone in your family? (ageMe + ageBrother + ageMom + ageDad)*
- *Is your brother older then you? (ageBrother > ageMe)*
- *Greater then and less than return Boolean results (true or false) in most programming languages*

Functions

Definition:

Functions group all the steps of a complex action into one command (e.g., brush teeth). Functions are especially useful for defining a sequence of commands that can be reused, for example how to turn a robot sideways by 90 degrees. The creation of functions is based on pattern recognition, abstraction and logical thinking.

Examples:

- *Function: Brush Teeth (involves turning on the water, putting toothpaste on toothbrush, etc.)*
- *Function: Draw a Square (draw a straight line of 3 cm, turn 90°, draw a straight line of 3 cm , turn 90°, draw a straight line of 3 cm, turn 90°, draw a straight line or 3 cm, turn 90°)*
- *Function: Header style <h1 style="color:Gray;">*

Inputs and Outputs

Definition:

Input and **output** (often abbreviated as I/O) is the communication that occurs between humans and computers, computers and other computers, among processes within a computer, or between a computer/robot and its environment. Humans interact with computers using devices called peripherals. When computers interact with other computers it is often over networks such as the internet. I/O occurs at all levels of the computer's operation.

Examples:

- *Humans provide input into computers using peripherals such as a keyboard, mouse, webcam, microphone, etc.*
- *Computers provide output using speakers, a screen, printers, etc.*
- *Robots use input devices such as sensors (light sensor, temperature sensor, etc.), buttons, etc.*
- *Robots' output devices include motors, speakers, actuators, lights, etc.*

1.7 Definitions: Computational Thinking Approaches

Unplugged

Definition:

Computational Thinking does not need a computer! Many activities that develop computational thinking can be done "**unplugged**," meaning away from computers. Playing games with rules, doing logic puzzles and creating and following recipes are all ways that Computational Thinking can be done "unplugged."

Examples:

- *Paper programming*
- *Drawing maps or completing mazes*
- *Playing games where one person tells someone else what to do or where to go*
- *Completing Truth Table (AKA liar and truth teller) logic games*

Tinkering

Definition:

In the context of CT, **tinkering** is about exploring code in a way that is playful and experimental. Tinkering involves changing things about a code to see what happens using code that someone else has created. Tinkering is important because it develops creative thinking and risk taking. It is also an effective way to begin to learn how a computer code works through cause and effect. Tinkering aligns with the ***USE*** phase in the Lee et al. (2011) Use-Modify-Create CT learning progression.

Examples:

- *Modifying the colours of clothing on a character in Scratch*
- *Adjusting the code to display your initials on the LED matrix in MakeCode*
- *Modifying the code for a robot to move in a new way*

Two girls "tinkering" with block code in the MakeCode coding environment.

Reusing and Remixing

Definition:

Reusing means taking pieces of code created by others and using it to solve a problem or meet a need, rather than creating it from scratch. **Remixing** involves putting together or 'mashing up' code for video, sound, text, etc. created by other people to make something new and original. Reusing and remixing aligns with the ***MODIFY*** phase in the Lee et al. (2011) Use-Modify-Create CT learning progression.

Examples:

- *Using images available through the Scratch library*
- *Taking a project that someone else did in Scratch and changing it to make it your own*
- *Reusing a function that someone else created (e.g., On Start, display temp)*

Making

Definition:

In the context of CT, **making** involves writing code from beginning to end, without using an existing code. This typically comes after familiarity with the language of a coding program and experience with approaches such as tinkering, reusing and remixing. Making aligns with the ***CREATE*** phase in the Lee et al. (2011) Use-Modify-Create CT learning progression.

Examples:

- *Creating an avatar of yourself and drawing it in Scratch*
- *Designing and building a LEGO™ Mindstorms robot from the various pieces that come in a kit*
- *Creating and coding a wearable sensor-based device using a micro:bit*

let's talk science | parlons sciences