Effective Computational Thinking (CT) and Coding Instruction - A Teacher's Guide

Table of Contents

Contents

Motivation for this Guide

Coding and computational thinking (CT) learning expectations have been integrated into the Ontario Math grades 1-8 (2020), Math grade 9 (2021), Science and Technology grades 1-8 (2022), and Science grade 9 curricula (2022). Many other jurisdictions within Canada and around the world have redefined learning expectations to include elements of CT and coding. In 2021, the OECD (Organization for Economic Cooperation and Development) included a set of computational thinking questions into the well known PISA (Programme for International Student Assessment) mathematics assessment.

Computational Thinking (CT) has various definitions, but the basic idea involves student exploration of how they can be creative and solve problems by decomposing situations into fundamental steps using algorithms and generalizing possible solutions. An **algorithm** is a precise set of steps that describe a process. **Coding** involves the use of a programming language environment to implement algorithms so learners can be creative and explore problem solving strategies.

The goal of this guide is to provide methods to help you, as an educator, deliver effective instruction in computational thinking and coding across the Science, Technology, Engineering, and Math (STEM) curriculum. The approaches described in this document are provided from many different information sources and are not limited to specific programming languages or grade levels. Following each teaching method, there will be a short explanation of how it might look in the classroom.

Ontario Curriculum Connections

The Grades 1-8 math curriculum guidance explains that "**coding can be incorporated across all strands and provides students with opportunities to apply and extend their math thinking, reasoning, and communicating**"[1] The Grades 1-8 Science and Technology curriculum includes a focus on “**Coding and the Impact of Coding and Emerging Technologies**”[2].

Coding is more than simply an alternative tool that can be used alongside more traditional tools such as paper and pencil, calculators, and scientific instruments (e.g., microscopes, measurement tools). Coding education is now embedded within the learning experiences of all students in Ontario. This guide will reference some simple examples of how coding pedagogical techniques could be used in the context of STEM activities in the classroom.

Learner Readiness - An Exploration of Universal Design for Learning

Students enter the classroom with a diverse set of skills and knowledge related to their understanding of coding. The pedagogical techniques explored in this document should be considered within the context of addressing this diversity and fostering an inclusive learning environment using the **Universal Design for Learning (UDL)** framework.

The Three (3) Principles of UDL are

- Engagement

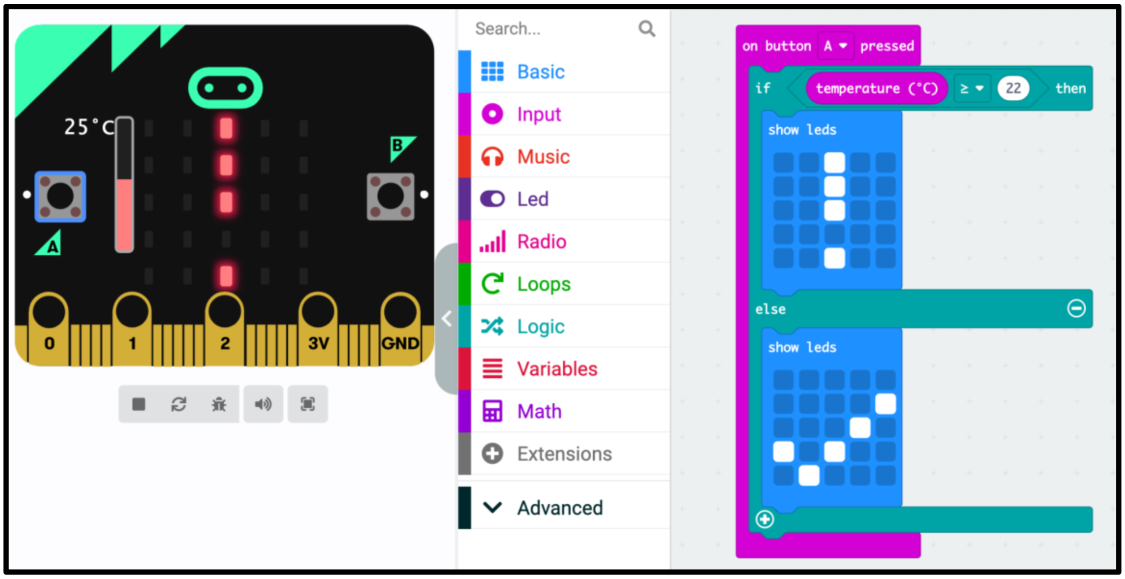
[1] https://www.dcp.edu.gov.on.ca/en/curriculum/elementary-mathematics/context/the-strands-in-the-mathematics-curriculum#strand-c-algebra
[2] [https://www.dcp.edu.gov.on.ca/en/sci-tech-key-changes/curriculum-context

- Representation
- Expression and Action

Engagement

This principle reflects the fact that learners have varied interests and their interests can change over time. Computational thinking and coding activities can be readily designed to engage a wide variety of student interests. Some examples of ***engagement*** techniques for coding education include

- Foster a sense of exploration
  - Encourage students to use coding to create images, tell stories, solve problems, control devices (**physical computing**), and design and build interactive games.



```
on button A pressed
    if temperature (°C) ≥ 22 then
        show leds
    else
        show leds
```

Figure 1: Block-based coding example

  - Sample code that can be downloaded to a micro:bit (physical computing) and manipulated/built on by students
- Model effective communication of new concepts or concept reviews through **live coding** (think-aloud programming).
  - **Live coding** is a demonstration of the act of coding, in person and live in front of a class. Teachers should encourage class discussions as the coded artifact is created. This can be an effective way to model problem solving, new computational concepts, and effective **debugging** techniques.
- Use **guided exploration** techniques such as **Use-Modify-Create** mode or, the more detailed, **PRIMM** model to start with existing coded artifacts to focus on understanding and code reading skills and progress toward student code writing skills.
- Allow for student choice during project based learning activities so they can explore their interests and foster culturally relevant and meaningful tasks.
- Reduce common pitfalls as learners develop their coding and problem solving skills.
  - There are many well known **misconceptions** in computing. Teachers should be prepared to explore the most common misconceptions with students. Students can become overwhelmed with errors while they learn to code. To help reduce these early barriers to success teachers can implement **Parsons Problems**. This technique involves providing all of the elements for a

successful program and the students focus on the problem solving task by arranging the code statements or code blocks.

- Encourage **collaboration** such as **pair programming** to support student success with coding.
- Chunk the learning into manageable units to support a sense of accomplishment for learners as they reflect on the concepts from each new activity.
- Set clear expectations that everyone in the classroom is able to be successful and everyone can share their knowledge with others as coding is best performed in a collaborative manner.

Representation

As in other subject areas, educators and learners develop a vocabulary and methods of knowledge representation. Reading, modifying, and writing programs are just one component of representing computational thinking and coding skills. Some examples of ***representation*** techniques for coding education include

- Model effective planning and design methods for coding projects including **pseudocode** and **flowcharts** for algorithms and **code tracing** of **variables**.
- Ensure coding environments are fully **accessible** for all learners. It is important that coding tools have the ability to adjust the text size or block sizes. Many coding tools will also use colours to indicate the type of error or the type of coding block. Primary students and English Language Learners (ELL) would be more successful if the amount of reading is reduced and the coding environment provides limited options to reduce the cognitive load for the learner. There are also flexible language setting options available in many of the coding environments referenced in this document.
- Provide timely **descriptive feedback**. Some digital tools will provide timely feedback, but teachers should model effective feedback and debugging skills in class discussions and individual student assistance sessions.
- Model effective **computing vocabulary** within class discussions and individual student assistance sessions.
  - e.g., debugging, test, execute/run, variable, operator, expression (see **glossary**)
- Model proper terminology when discussing concepts and providing structured learning materials
  - e.g., store the value 5 in the variable age, update the value stored in the variable by 2, repeat the statements until a condition is met or a specific number of times
- Educators have a responsibility to ensure that teaching and learning in the areas of coding and computational thinking are culturally responsive and relevant, making sure that all students are included in coding opportunities through carefully-selected, relevant learning tasks. Students should use “STEM and CS [coding] to empower themselves and their communities”(Madkins, 2020), and educators should acknowledge and address that many algorithms are not without bias and racism.

Action and Expression

Learners express their understanding and knowledge of coding using various methods including physical, written, and oral expressions of their ideas. Teachers can provide multiple methods for learners to express their ideas through code. Some examples of ***action and expression*** include:

- Provide scaffolded coding activities including starter projects that include working code elements. Learners can use these scaffolds to read and understand existing code elements and communicate their modification using new code blocks/statements and associated comments explaining their thinking.

- Jointly design project goals and **assessment templates** (eg. single point rubrics) with students.
- Provide opportunities for students to **journal** their coding progress. Include student reflection opportunities throughout their learning.
- Teachers should provide assessment opportunities for students using observation, conversations, and artifacts (products). (See the **Assessment Strategies** section of this document)

What Does Coding and Computational Thinking Look Like?

Selecting a programming language can be a challenging task and educators should consider the context of the coding activity and the students' prior experience with coding and ***computational*** thinking (CT). "Coding can include a combination of **pseudocode**, **block-based coding** programs, and **text-based** coding programs.". (Grade 1 Mathematics Key Concept[3])

Pseudocode is an informal method of designing a set of computational operations. Whether you decide to use block-based or text-based coding tools, traditional coding concepts can all be categorized as either: sequential, selection, or repetition tasks.

Block-based coding environments are excellent tools to explore CT and coding for Grades 1-8. Programs are usually called **scripts** and they consist of a collection of blocks or puzzle pieces that define the operations or steps involved in the coding task. Writing or modifying programs involves the dragging and dropping of various blocks to achieve the desired result. As students modify programs they will encounter errors, but the types of errors encountered with block-based coding environments are usually restricted to logic errors instead of language violation error.

Code with Syntax Error	**Corrected Code**

`# Compute the area of a rectangle`
`width = 10`
`width = 20`
`area = width X height`
`print (area)`
	`# Compute the area of a rectangle`
	`width = 10`
	`width = 20`
	`area = width * height`
	`print (area)`

Figure 2: Common Coding Error Example

A programming language violation error is usually referred to as a **syntax error**. The example code in Figure 2 involves the incorrect use of the letter X to perform a multiplication operation. Python, and many other text-based programming languages, require the use of the asterisk (*) character to indicate a multiplication operator. As we can see in this example coding requires a high level of precision as computers are not able to determine the intent or purpose of the program.

ScratchJr is an excellent block-based coding environment that provides these fundamental components in a very accessible manner for primary level students. With ScratchJr, students: "can program their own interactive stories and games. In the process, they learn to solve problems, design projects, and express themselves creatively on the computer." (Source: ScratchJr website).

[3] https://www.dcp.edu.gov.on.ca/en/curriculum/elementary-mathematics/grades/g1-math/strand-c/c3

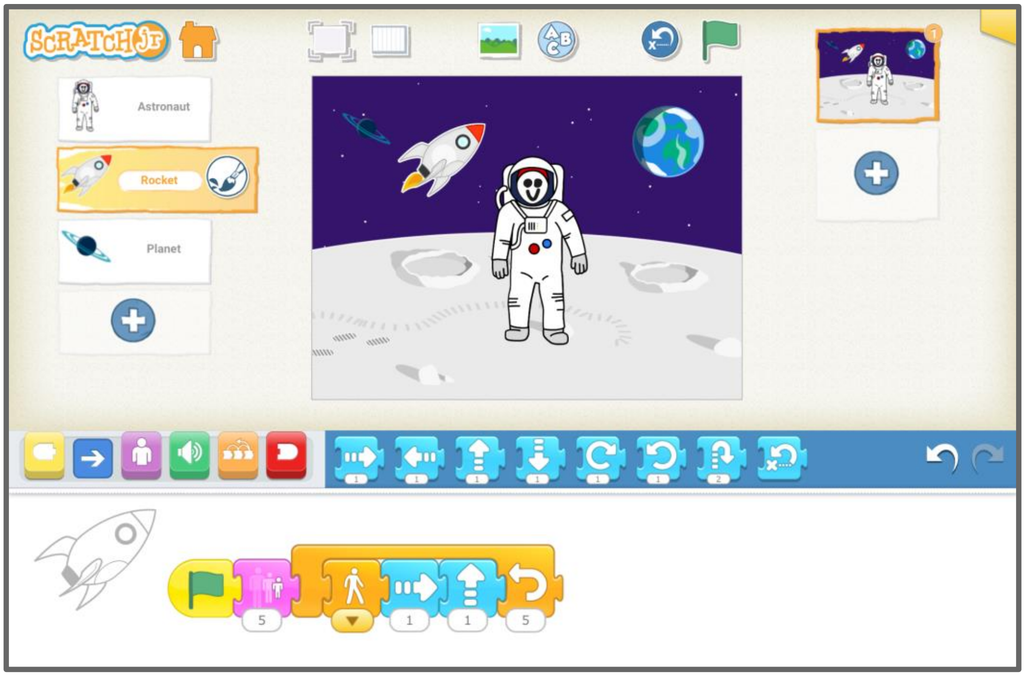
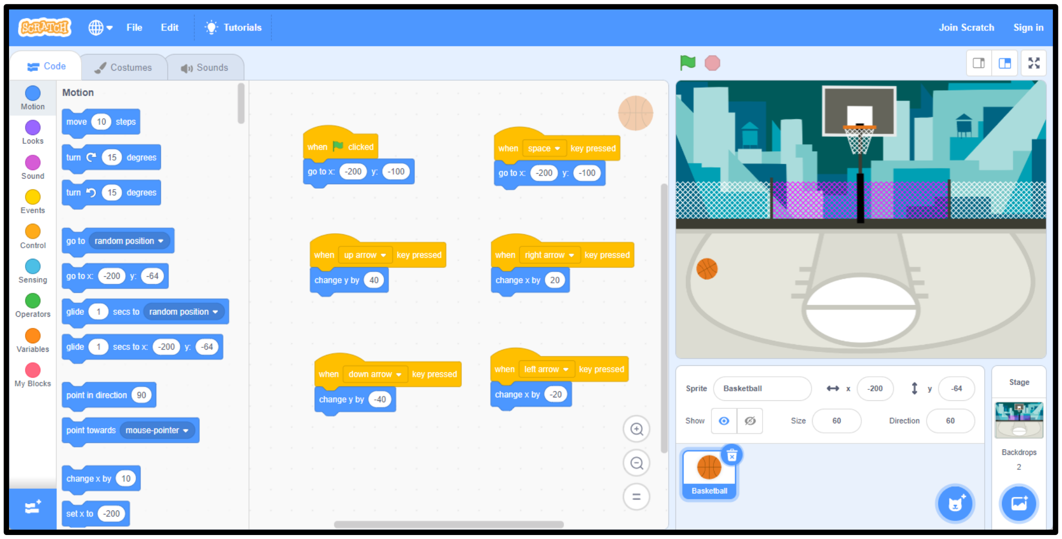


Figure 3: ScratchJr (Block-based Coding) Example

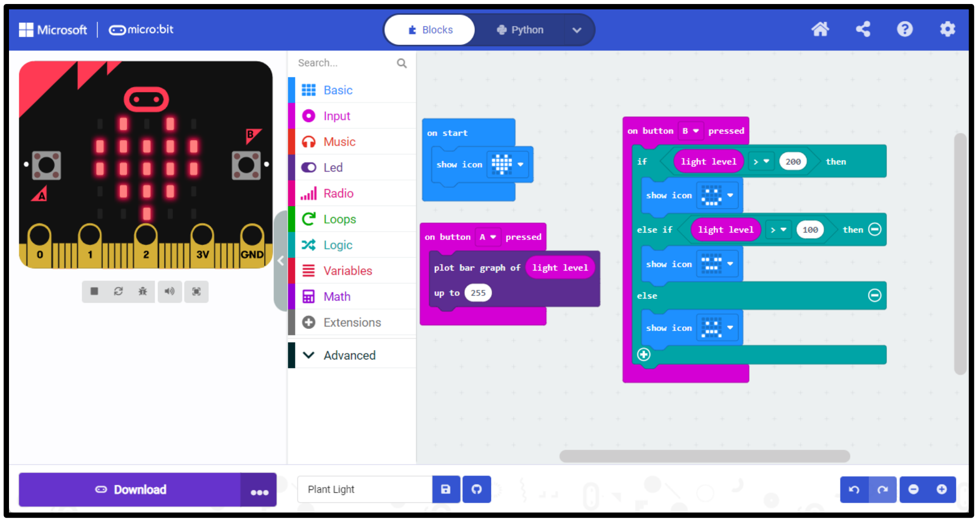
Scratch provides an expanded set of features to enable additional computational problem solving and coding skills. Scratch introduces new concepts such as concurrent events (using broadcast blocks), variables, and custom blocks (functions or subprograms).



```
when ⚑ clicked
go to x: -200 y: -100

when space key pressed
go to x: -200 y: -100

when up arrow key pressed
change y by 40

when right arrow key pressed
change x by 20

when down arrow key pressed
change y by -40

when left arrow key pressed
change x by -20
```

Figure 4: Scratch (Block-based Coding) Example

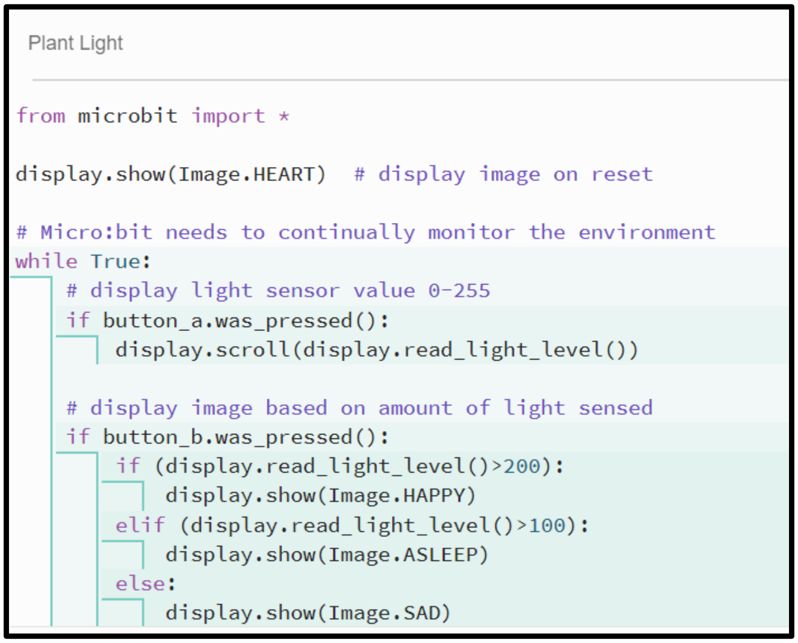
Microsoft MakeCode provides similar features as Scratch with the addition of physical computing programs using a microcontroller such as Micro:bit. MakeCode provides learners with the ability to develop their projects using both block-based and text-based programming languages (Python and Javascript).



```
on start
    show icon

on button A pressed
    plot bar graph of light level
    up to 255

on button B pressed
    if light level > 200 then
        show icon
    else if light level > 100 then
        show icon
    else
        show icon
```

Figure 5: Microsoft MakeCode Micro:bit (Block-based Coding) Example

Text-based coding environments provide students with additional options to explore the curriculum. The most commonly used text-based coding language is **Python**. Using text-based coding requires learners to have novice level **keyboarding skills**.


Plant Light

```
from microbit import *

display.show(Image.HEART)  # display image on reset

# Micro:bit needs to continually monitor the environment
while True:
    # display light sensor value 0-255
    if button_a.was_pressed():
        display.scroll(display.read_light_level())

    # display image based on amount of light sensed
    if button_b.was_pressed():
        if (display.read_light_level()>200):
            display.show(Image.HAPPY)
        elif (display.read_light_level()>100):
            display.show(Image.ASLEEP)
        else:
            display.show(Image.SAD)
```

Figure 6: Micro:bit (text-based Coding - Python)

Physical Computing can take many forms from unplugged activities to motors and more.

Unplugged and Physical Computing Activities

The CS Unplugged[4] activities are excellent resources for teachers to consider using with their students to explore coding concepts. The activities do not require the use of a computer, but they support student learning of how information is represented by computers and how computers can be used to aid in problem solving

[4] CS Unplugged (https://www.csunplugged.org/en/ / https://www.csunplugged.org/fr/)

tasks. CS Unplugged activities use role playing and concrete artifacts in support of student engagement. Students can use the provided activity cards to explore algorithms. If there are classroom resources such as physical robot devices (e.g., Ozobot, Sphere, Dash, etc) available, the algorithms can be tested using actual devices.

"Physical computing covers the design and realization of interactive objects and installations and allows students to develop concrete, tangible products of the real world that arise from the learners' imagination. This way, constructionist learning is raised to a level that enables students to gain haptic experience and thereby concretizes the virtual." (Przybylla/Romeike) Bringing physical devices into CT and coding activities provides an opportunity for students to interact directly with the code as they create programs that sense the world around them or change their environment. Some commonly used devices for physical computing other than robots include the Micro:bit and Phidgets.

The micro:bit is one example of a microcontroller. Microcontrollers are similar to computers, they have a processor, and generally have different inputs and outputs of varying complexity. The micro:bit from the BBC has become a commonly used microcontroller due to cost, ease of use, and variety of applications. See the diagram below for a quick breakdown of the different parts of the micro:bit.

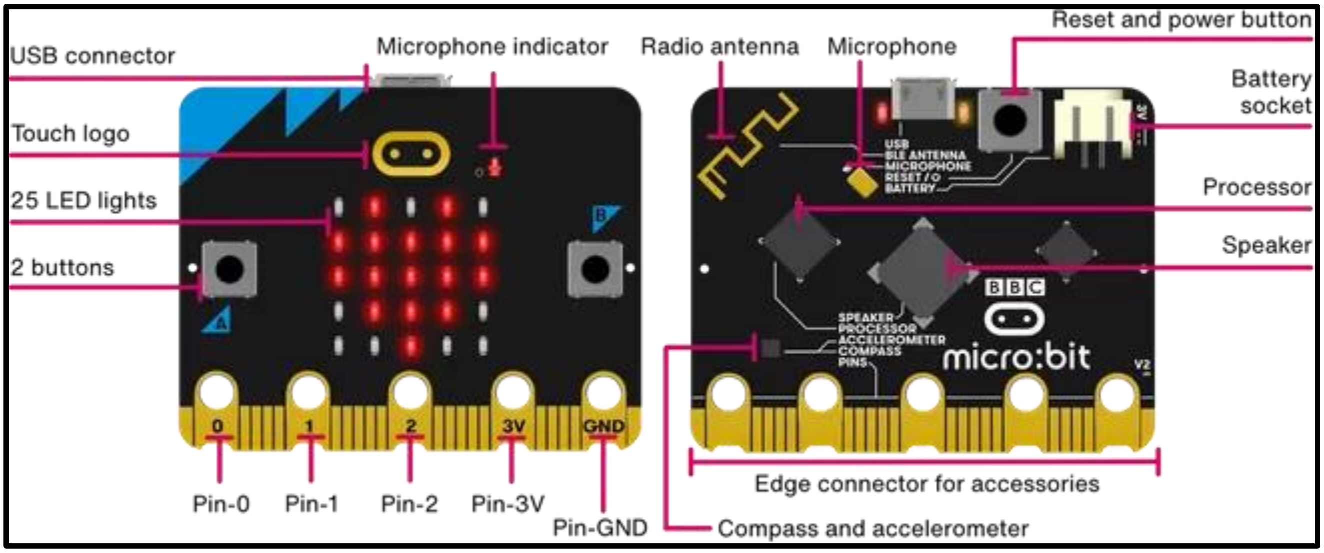


Figure 7: Micro:bit - Microcontroller

In addition to the built-in capabilities of the micro:bit, Pin 0, Pin 1, or Pin 2 can be used to connect to other external devices such as LEDs (Light Emitting Diodes) or servos.

Micro:bits and many other boards can be combined with input sensors to measure different values, for example, moisture levels, touch, sound intensity, or CO_2 levels. Additionally, the boards can provide different outputs, for example turning a water pump on, or turning a servo or motor.

Scaffolding Learning to Code

Copy-Code/Modify/Extend Activities

A highly structured coding activity is often known as a **coding tutorial**. This approach often involves a set of written or oral instructions to be followed precisely to create a working program. Each learner is able to recreate the same program if the instructions are followed well, but there is often minimal knowledge retained by learners along the way unless there are opportunities to consolidate the problem solving and coding strategies that were used to design the program.

Activity 1 - Exploring Concurrent Events (Scratch) - Copy/Modify/Extend

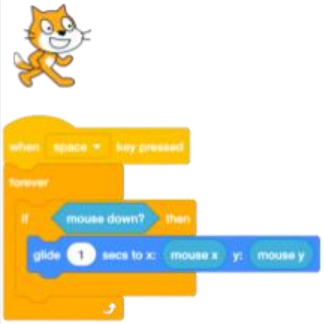
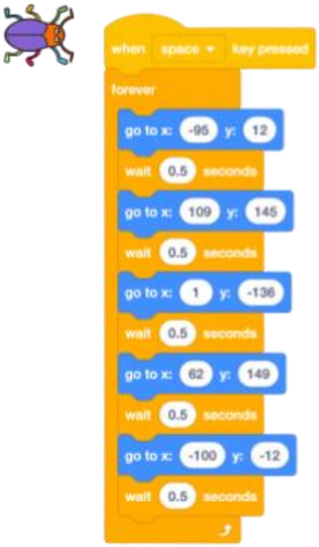
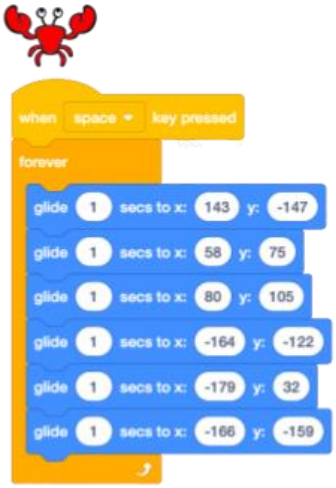
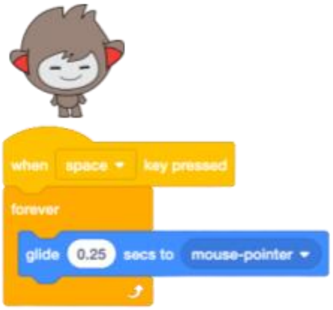
Coding tutorials or **copy code** activities are effective initial opportunities for students to gain some success with coding. In the example below, students begin with a copy code activity to see different ways to have sprites move around the stage. They are prompted to observe how the sprites are moving around and to make connections back to the code. Students then can modify the code to help them apply some of their learning. Finally, students can go beyond the activity by choosing their own sprites for the letters of their name and animate the letters using the movement code that they've copied, to make a dance-party-like animation of their name.

Example[5]:

[5] Source document: https://docs.google.com/drawings/d/1D0Th8Q7VyZnP2TZrLE2_m8xKRrVwLw6Ke0slYLic2wE/edit?usp=sharing

Stage 1: Copy Code/Replicate

Use the sprites below and copy the given code into each sprite. Run the code and observe the code, how do they move, do they teleport or glide? How can you tell by looking at the code?

 when space ▾ key pressed forever if mouse down? then glide 1 secs to x: mouse x y: mouse y	 when space ▾ key pressed forever go to x: -95 y: 12 wait 0.5 seconds go to x: 109 y: 145 wait 0.5 seconds go to x: 1 y: -136 wait 0.5 seconds go to x: 62 y: 149 wait 0.5 seconds go to x: -100 y: -12 wait 0.5 seconds	 when space ▾ key pressed forever glide 1 secs to x: 143 y: -147 glide 1 secs to x: 58 y: 75 glide 1 secs to x: 80 y: 105 glide 1 secs to x: -164 y: -122 glide 1 secs to x: -179 y: 32 glide 1 secs to x: -166 y: -159	 when space ▾ key pressed forever glide 0.25 secs to mouse-pointer ▾
Observe and record how this sprite moves:	Observe and record how this sprite moves:	Observe and record how this sprite moves:	Observe and record how this sprite moves:

Stage 2: Modify

Now change the code for the crab and the beetle so that they don't just move around with a set of instructions, but they respond to the user's actions, like scratch the cat and Nano do.

Stage 3: Extend

Create a new project where you code the letters to move around either independently or in response to user input like a animated letter dance party! Bonus - add in some music to go with the animations.

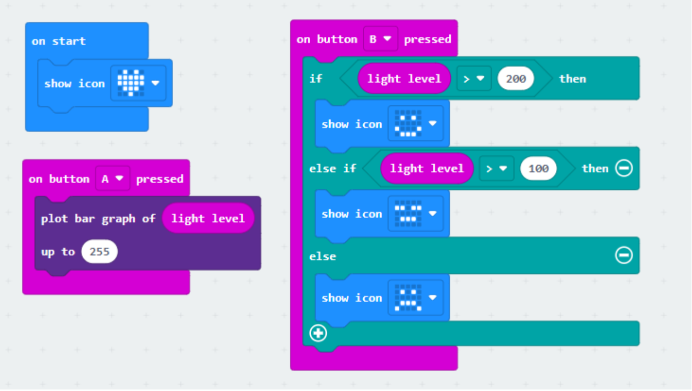
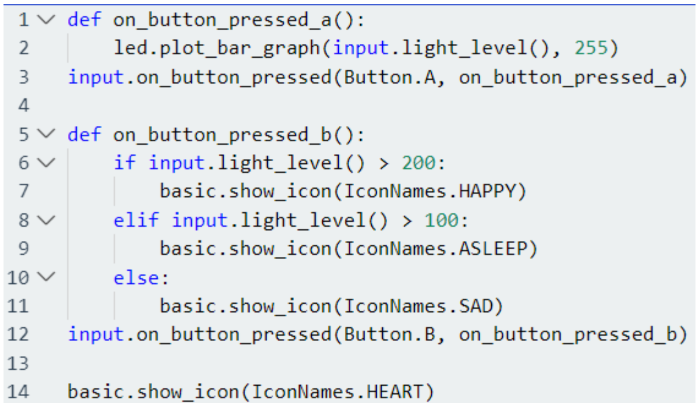
Activity 2 - Exploring Light and Plant Growth (Micro:bit) - Copy/Modify/Extend

An example of copy coding in Python and Science & Technology is provided below[6]. The activity could be implemented using Blocks or a Python coding environment in Make Code or the Micro:bit Editor for Python.

[6] Source document: https://docs.google.com/drawings/d/1QrHVJWE5qAItXmgvZZ5YmN5ppYKIXYyURguY6W5hhZY/edit?usp=sharing

As students progress in the activity, students are able to practice detecting and adjusting their design for different levels within an abiotic factor like brightness and design the output. The complexity of the coding can be increased by having students detect and design outputs for combined abiotic factors.

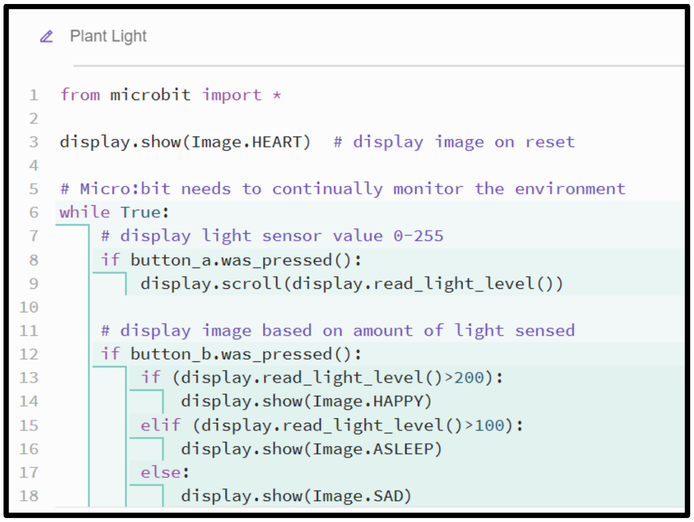
Microsoft MakeCode - Micro:bit using Blocks or Python (https://makecode.microbit.org/)

<table>
<tr><th>Blocks</th><th>Python</th></tr>
<tr><td>
</td><td>
<pre>
def on_button_pressed_a():
    led.plot_bar_graph(input.light_level(), 255)
input.on_button_pressed(Button.A, on_button_pressed_a)

def on_button_pressed_b():
    if input.light_level() > 200:
        basic.show_icon(IconNames.HAPPY)
    elif input.light_level() > 100:
        basic.show_icon(IconNames.ASLEEP)
    else:
        basic.show_icon(IconNames.SAD)
input.on_button_pressed(Button.B, on_button_pressed_b)

basic.show_icon(IconNames.HEART)
</pre></td></tr>
</table>

Micro:bit Editor - Python (https://python.microbit.org/v/3)

<table>
<tr><th>Python</th></tr>
<tr><td>
Plant Light
<pre>
from microbit import *

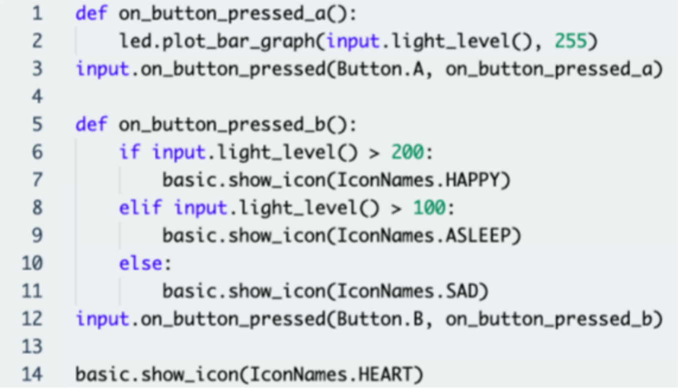
display.show(Image.HEART)   # display image on reset

# Micro:bit needs to continually monitor the environment
while True:
    # display light sensor value 0-255
    if button_a.was_pressed():
        display.scroll(display.read_light_level())

    # display image based on amount of light sensed
    if button_b.was_pressed():
        if (display.read_light_level()>200):
            display.show(Image.HAPPY)
        elif (display.read_light_level()>100):
            display.show(Image.ASLEEP)
        else:
            display.show(Image.SAD)
</pre></td></tr>
</table>

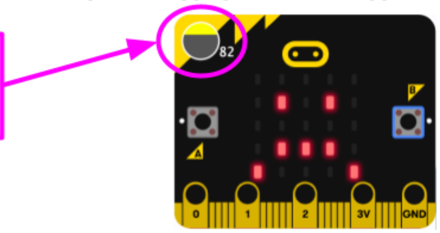
Stage 1: Copy Code/Replicate

Use the code below in the makecode micro:bit python coding development environment. The design of the code it to use the built-in light sensor on the micro:bit to detect the brightness level around a window plant and display output of three different 'faces' depending on the light level.



```
def on_button_pressed_a():
    led.plot_bar_graph(input.light_level(), 255)
input.on_button_pressed(Button.A, on_button_pressed_a)

def on_button_pressed_b():
    if input.light_level() > 200:
        basic.show_icon(IconNames.HAPPY)
    elif input.light_level() > 100:
        basic.show_icon(IconNames.ASLEEP)
    else:
        basic.show_icon(IconNames.SAD)
input.on_button_pressed(Button.B, on_button_pressed_b)

basic.show_icon(IconNames.HEART)
```

Explore what the code does by changing the icons displayed as feedback (`IconNames.SAD` for example). Try to adjust the different ranges of the conditional "If…elif…else" to see how you can change the ranges. Make sure to test your code by pressing and dragging the sensor toggle in Makecode:



Sensor Toggle in micro:bit simulator, for testing the code.

Stage 2: Modify

Now change the code by adding in more `elif` conditions to allow for finer increments and better feedback for the user.

Stage 3: Extend

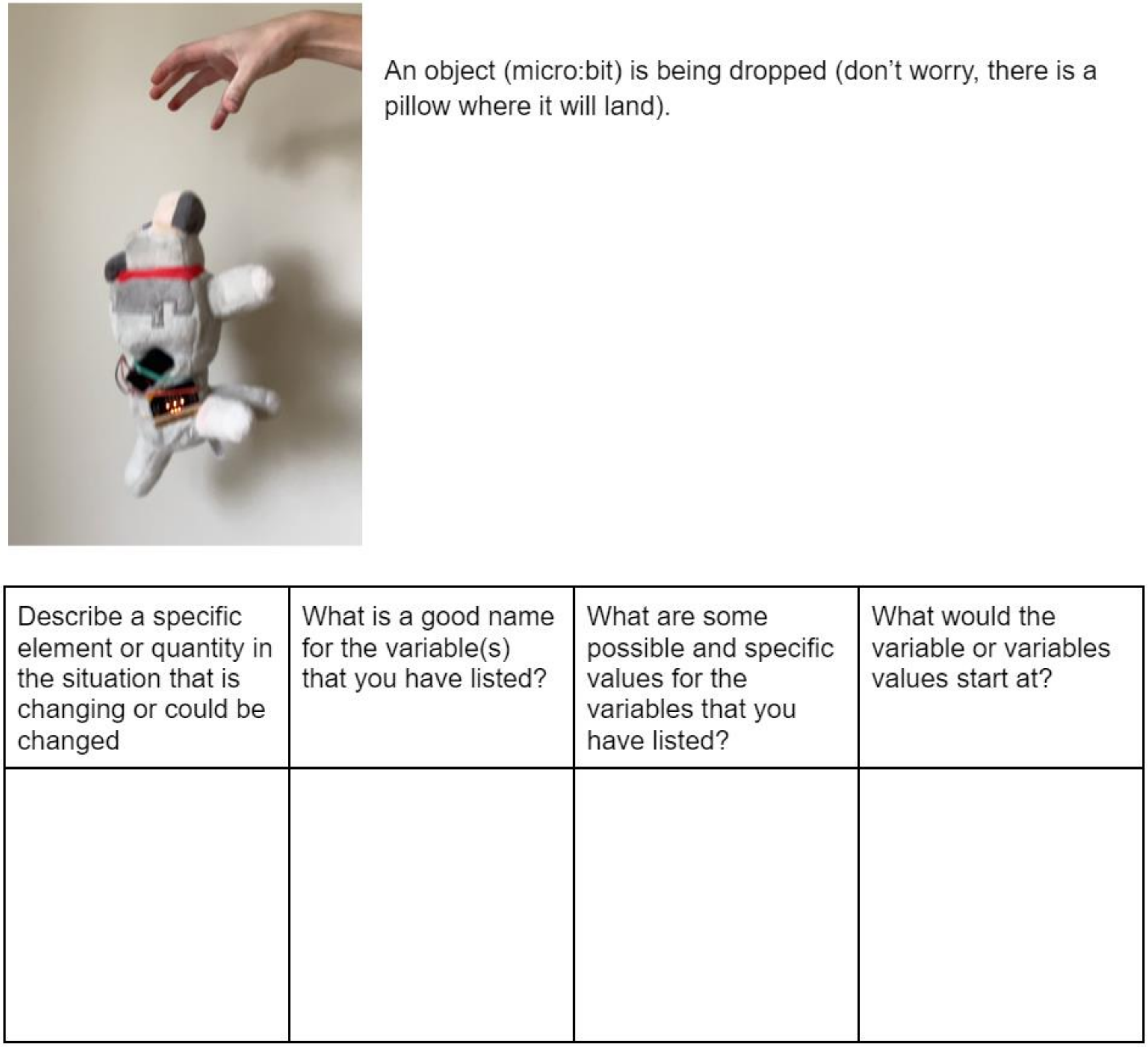
Create a new project where you detect and provide feedback for temperature; go further and provide feedback for COMBINED temperature AND brightness, OR build your own moisture sensor with two nails and detect the moisture of a potted plant.

https://makecode.microbit.org/_Keg7Jg7deXu4

Story variables

Story variables can be used to support the Initiating and Planning phase of the Engineering Design Process for coding projects. Phil Bagge has developed the story variable approach to introduce the concept of code **variables** within the context of stories.

Example of Story Variables in Science and Technology[7]



An object (micro:bit) is being dropped (don't worry, there is a pillow where it will land).

Describe a specific element or quantity in the situation that is changing or could be changed	What is a good name for the variable(s) that you have listed?	What are some possible and specific values for the variables that you have listed?	What would the variable or variables values start at?

In the Science and Technology themed example, students may think of values like the weight of the object that the micro:bit is attached to, the height from which the object is being dropped, the velocity of the object as it falls or the direction in which the object is moving.

[7] Source document: https://docs.google.com/document/d/11EjxPT2OtR3zYBunfU8yRiBg_w03uef8wRCEPnCNSjw/edit?usp=sharing

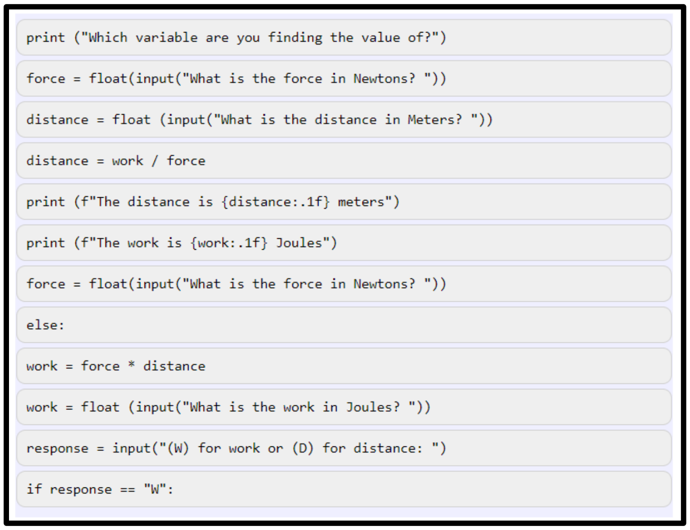
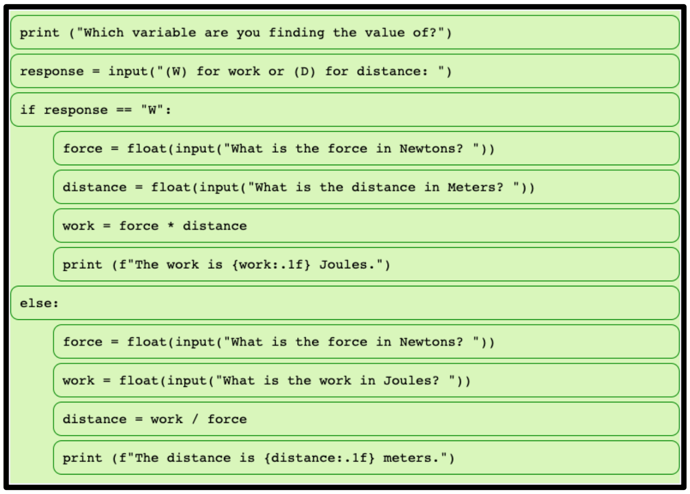
Guided Tasks

Parsons Problems

Parsons problems are used to assess student understanding of code without regard to language syntax. Students are provided with all of the required code blocks or statements to solve a challenge, but not in the proper sequence. Parsons problems can be implemented in many forms including programming language statements, pseudocode, or blocks.

Parsons problems are an assessment tool that eliminates the need for students to worry about programming language syntax or coding blocks. Students can simply focus on the problem solving task.

Here is an example[8] of a Parsons problem using Python. In the example, it uses the Work = Force x distance formula and allows students to order the code and Python will determine the result.

Parsons Problem	**Solution**
	

```
print ("Which variable are you finding the value of?")
force = float(input("What is the force in Newtons? "))
distance = float (input("What is the distance in Meters? "))
distance = work / force
print (f"The distance is {distance:.1f} meters")
print (f"The work is {work:.1f} Joules")
force = float(input("What is the force in Newtons? "))
else:
work = force * distance
work = float (input("What is the work in Joules? "))
response = input("(W) for work or (D) for distance: ")
if response == "W":
```

```
print ("Which variable are you finding the value of?")
response = input("(W) for work or (D) for distance: ")
if response == "W":
    force = float(input("What is the force in Newtons? "))
    distance = float(input("What is the distance in Meters? "))
    work = force * distance
    print (f"The work is {work:.1f} Joules.")
else:
    force = float(input("What is the force in Newtons? "))
    work = float(input("What is the work in Joules? "))
    distance = work / force
    print (f"The distance is {distance:.1f} meters.")
```

Figure 8: Parsons Problem - Structures and Mechanisms

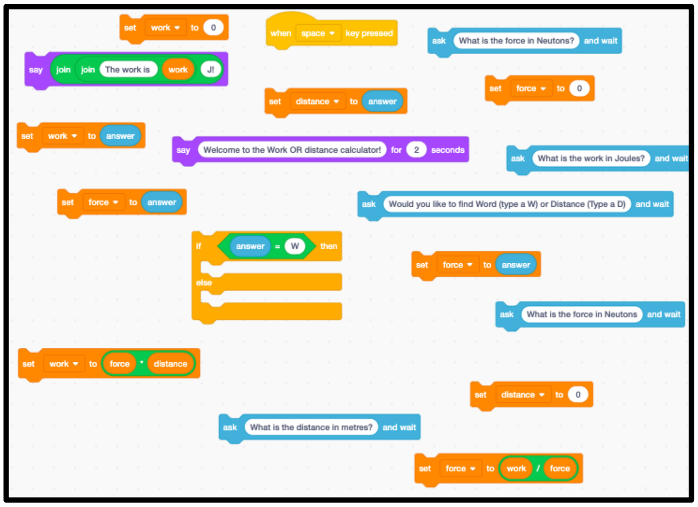
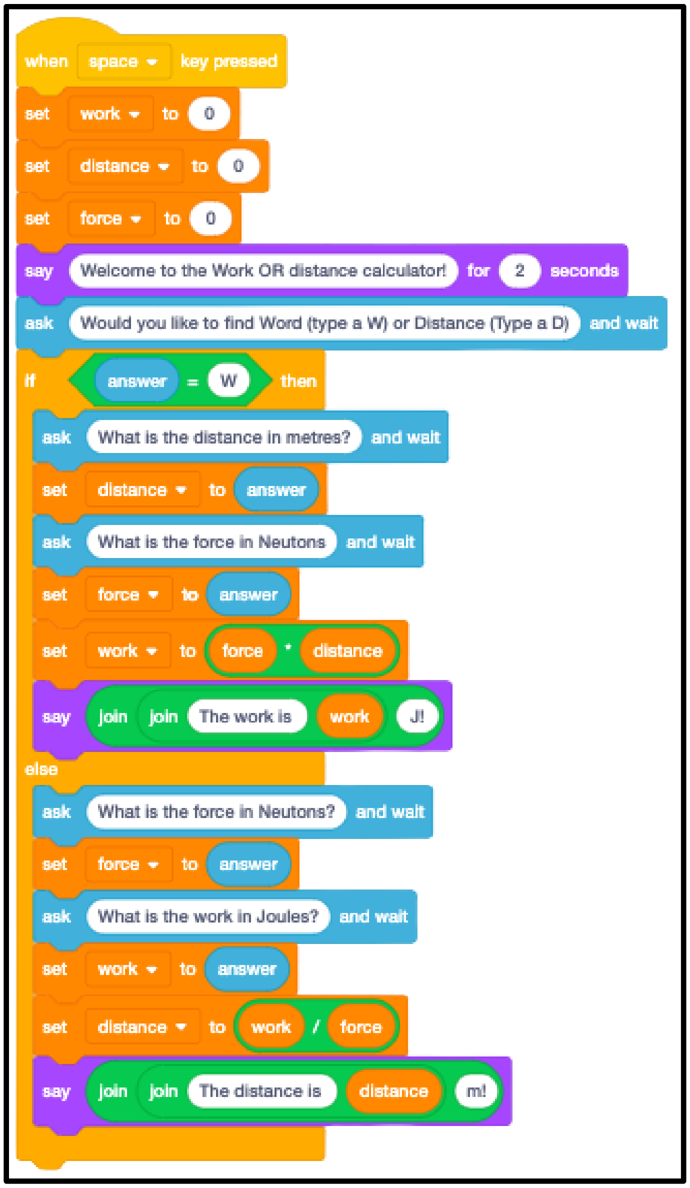
Parsons problems can also be created for **block-based languages** and they can provide students with immediate feedback.

In Scratch, Parsons problems can be made within Scratch with the blocks provided to students so that they can arrange the blocks, test, and then adjust their code in rapid succession. In the sample provided students can use the disabled, but once-functioning code to work again.

Example of Parsons problems in Science and Technology

[8] Source: http://parsons.problemsolving.io/puzzle/563342add2314102815a97a9b05595e7

In this example we use a Scratch script to calculate the work or distance based on the input from the user. No additional blocks are needed, and no extra blocks are included.

Parsons Problem	Parsons Problem Solution
 set work to 0 when space key pressed ask What is the force in Neutons? and wait say join join The work is work J! set force to 0 set distance to answer set work to answer say Welcome to the Work OR distance calculator! for 2 seconds ask What is the work in Joules? and wait set force to answer ask Would you like to find Word (type a W) or Distance (Type a D) and wait if answer = W then else set force to answer ask What is the force in Neutons and wait set work to force * distance set distance to 0 ask What is the distance in metres? and wait set force to work / force	 when space key pressed set work to 0 set distance to 0 set force to 0 say Welcome to the Work OR distance calculator! for 2 seconds ask Would you like to find Word (type a W) or Distance (Type a D) and wait if answer = W then ask What is the distance in metres? and wait set distance to answer ask What is the force in Neutons and wait set force to answer set work to force * distance say join join The work is work J! else ask What is the force in Neutons? and wait set force to answer ask What is the work in Joules? and wait set work to answer set distance to work / force say join join The distance is distance m!

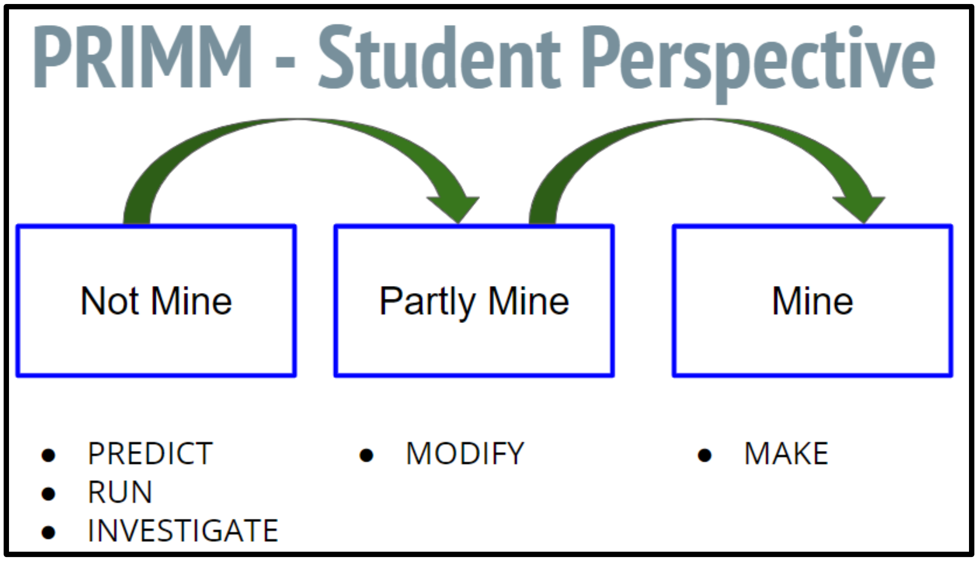
Example Scratch Parsons Problem Project: https://scratch.mit.edu/projects/753631554

PRIMM

PRIMM (Predict-Run-Investigate-Modify-Make) represents a guided discovery approach to learning where "classroom activities can be designed which involve predicting the output of code, code comprehension and gradually making new programs" (Sentance, 2020). PRIMM can be considered an elaboration of a more simplified **Copy-Code/Modify/Extend** approach discussed in the previous section of this document.

PRIMM starts with an analysis of existing code individually, in small groups, or as a whole class. The learners are asked to ***predict*** the results. This task engages the learner's mental model of computation and understanding of data structures and algorithms. There must be sufficient prior experience and learning with the coding environment for students to be well positioned to predict the outcome of the program. This is a low stakes activity where there could very well be varying interpretations of the intentions of the program.

The ***run*** phase can be completed by the teacher or by individual students once they have made their predictions. The teacher should then follow this coding project with a set of inquiry or **investigation** tasks or "what if scenarios…". These inquiry tasks can help consolidate the learner's understanding of the coding elements. The students are then provided with opportunities to explore open-ended projects by **modifying** or **making** new programs using some of the concepts introduced in the learning activity.



PRIMM - Student Perspective

Not Mine

Partly Mine

Mine

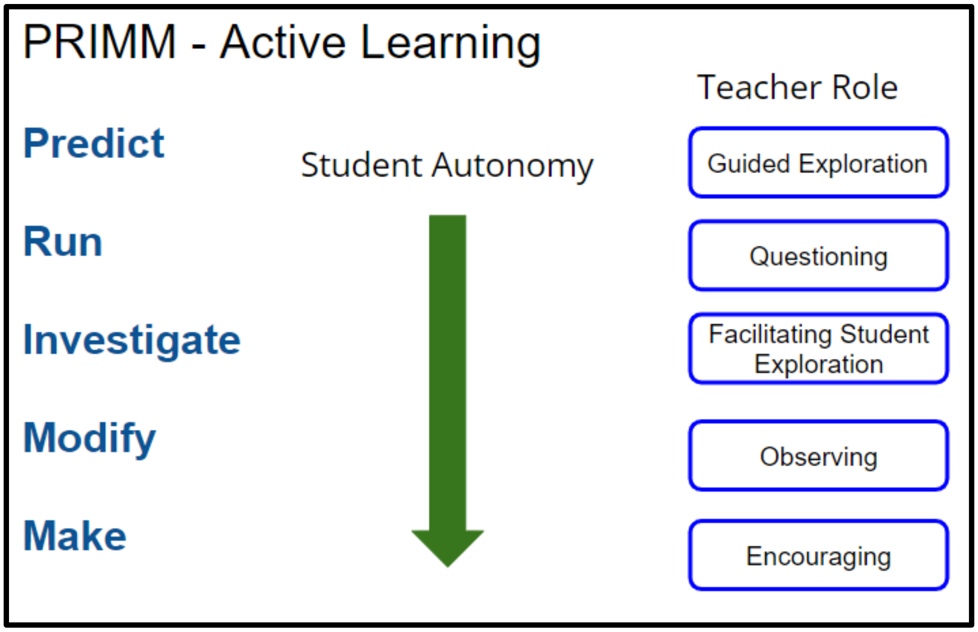
- PREDICT
- RUN
- INVESTIGATE

- MODIFY

- MAKE

Figure 9: Student Perspective of PRIMM Activities - Progression of Ownership

As shown in Figure 9, students will begin the PRIMM process by examining an existing program to predict its behavior. Students will then run the code to validate their understanding of the code. Students will then be provided with investigation opportunities to make modifications to the existing program. As students become more confident in their understanding of the computational concepts involved in the investigation they can plan and implement (code) a new project based on their interests.



PRIMM - Active Learning

Predict

Run

Investigate

Modify

Make

Student Autonomy

Teacher Role

Guided Exploration

Questioning

Facilitating Student Exploration

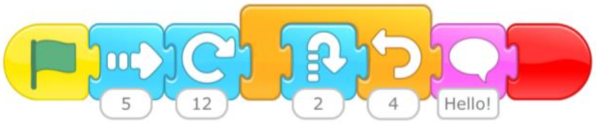
Observing

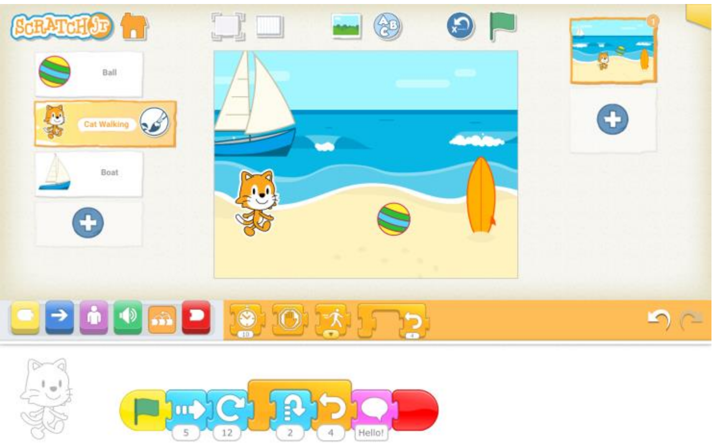
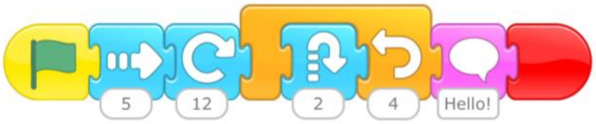
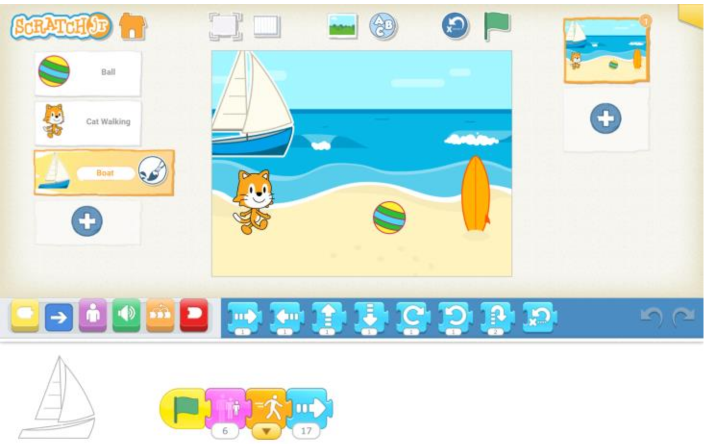
Encouraging

Figure 10: Teacher Role of PRIMM Activities

Figure 10 provides guidance for teachers as they prepare the initial "starter" project with guided exploration questions. Teachers will slowly transition from guided instruction to supporting and encouraging students as they plan and implement student-initiated coding projects.

Activity 3 - Exploring Movement (Scratch Jr) - PRIMM

PRIMM:	ScratchJR Example:	Students are…
Predict *What will this script do to the Scratch cat?*		- turn-and-talking with a partner - contributing to a whole-class conversation - writing or drawing what they think will happen

Run *Run the script and watch what happens.*		- running the code independently or with a partner - observing what happens when the code is run
Investigate *What block is making the sprite move forward? What is the result of the loop?*		- examining the code to identify the components (e.g., inputs, outputs, etc.) - identifying which block(s) cause which action(s)
Modify *Add a sprite or choose 1 of the other ones already present (Ball and Boat). Modify the code so that these sprites move.*		- modifying the code by adding, deleting, or altering what is already present in order to modify the input, output, etc.
Make *Create a ScratchJR project where the Cat interacts with another sprite.*		- creating a project where they apply the skills they have learned

Targeted Task - Misconceptions

Targeted tasks can be designed to expose potential misconceptions about writing code. These tasks should require a short duration to complete, and they should be accompanied by a set of inquiry questions about the misconception being investigated. For example, the sequence or order of coding statements or blocks is very important so students could be given tasks that involve the incorrect order and ask them to figure out how to correct the error by changing the order.

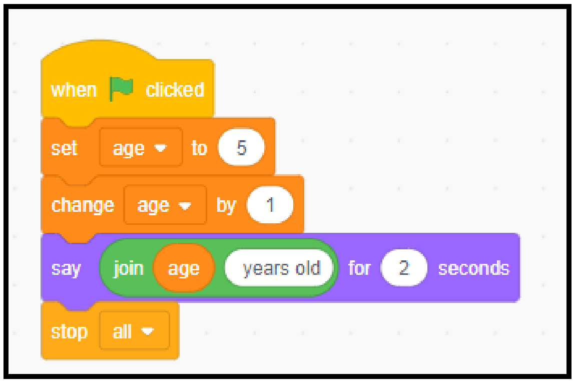
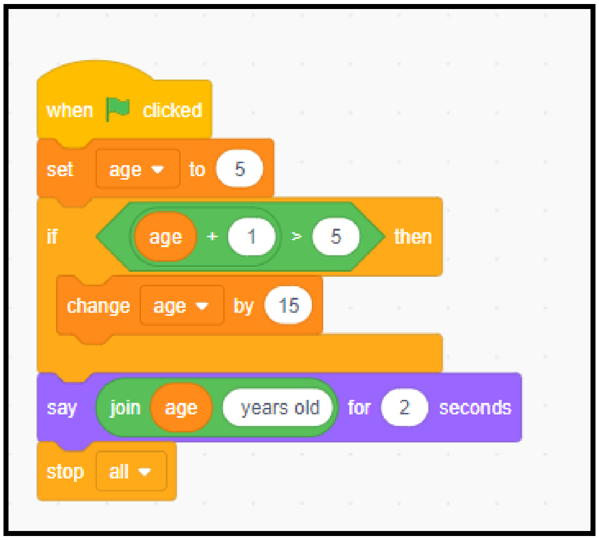
There are many common misconceptions formed by novice programmers.

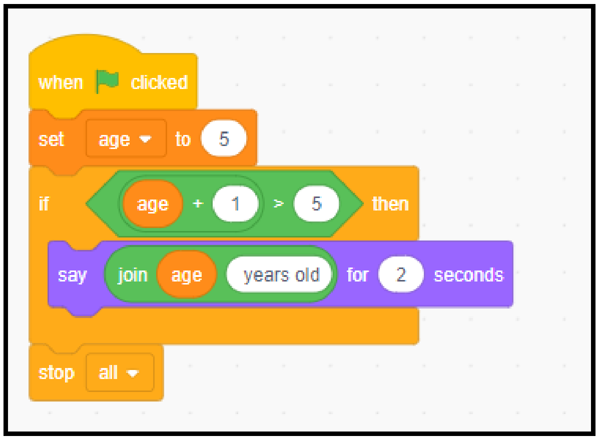
Variable Misconceptions

Variables are used to manage state during the execution of a program. Variables have an initial value, either implied or explicitly assigned. The value stored in the variable can either remain the same throughout the program or it could be modified.

Variable Misconception Example 1

When using Scratch students are often confused by the difference between a **set block**, **change by block**, and a **math operator block** when modifying or comparing values stored in variables.

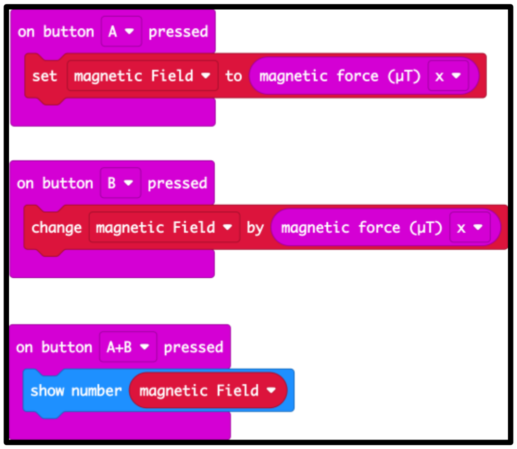
Scratch Script	What is the value stored in the variable **age**?
 when ⚑ clicked set age ▾ to 5 change age ▾ by 1 say join age years old for 2 seconds stop all ▾	6
 when ⚑ clicked set age ▾ to 5 if age + 1 > 5 then change age ▾ by 15 say join age years old for 2 seconds stop all ▾	20

	5

```
when ⚑ clicked
set age to 5
if age + 1 > 5 then
    say join age years old for 2 seconds
stop all
```

Variable Misconception Example 2

In a physical computing example, a program for a micro:bit was made where the student wants to determine the magnetic field strength of different objects and specific magnets, bar, u shaped, and neodymium disk magnets in terms of relative strength and positive or negative regions by area for the magnets. The sample below contains an example of the variable misconception. When the “A” button is pressed, the variable is correctly made equal to the magnetic field strength. When the “B” button is pressed, the variable has the value of the magnetic field added to the existing value stored in the variable.



```
on button A pressed
    set magnetic Field to magnetic force (µT) x

on button B pressed
    change magnetic Field by magnetic force (µT) x

on button A+B pressed
    show number magnetic Field
```

Communication

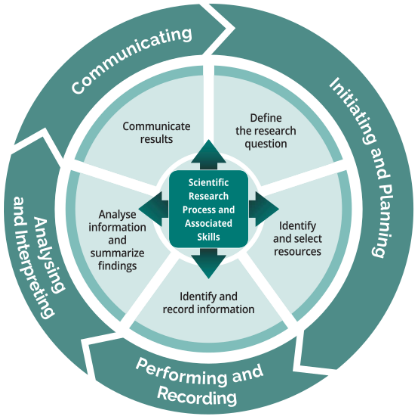
Communication occurs at many different levels in CT and coding. CT and coding can be explored as a **literacy** or language skill. There are fundamental differences between the less formal environment of human language and the precision required using programming languages. The use of block-based programming environments can provide a more accessible on-ramp, otherwise known as a low ceiling, to coding as the blocks represent actions and they help decrease the likelihood of minor coding errors that are likely with text coding environments and syntax errors.

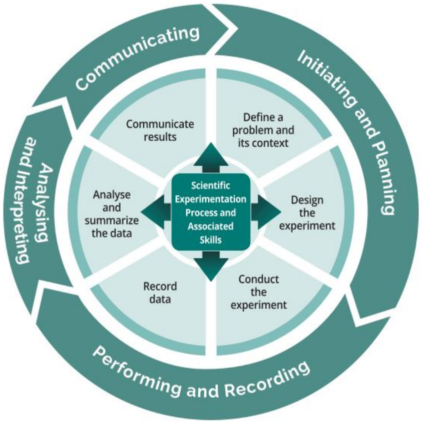
Educators can scaffold according to the needs of students through the introduction of only the necessary coding blocks/coding statements that would be required for the **activity**. Educators should provide many worked examples within the context of a creative task or problem solving activity before requesting students to attempt to write new programs of their own design.

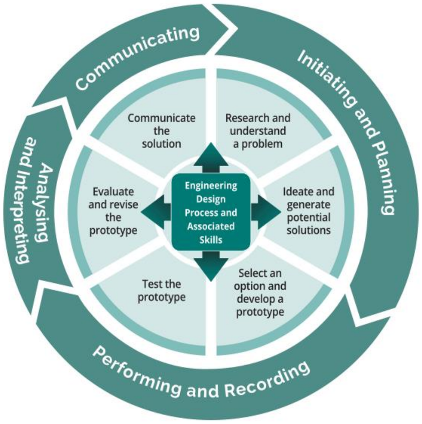
The Science and Technology Grades 1-8 curriculum in Ontario describes the **scientific research process**, the **scientific experiment process**, and the **engineering design process**[9].

When applying coding and computational thinking skills in the science and technology classroom consider what it looks like within the following phases:

- **initiating and planning** (e.g., asking questions, clarifying problems, planning procedures)
- **performing and recording** (e.g., following procedures, accessing information, recording observations and findings)
- **analysing and interpreting** (e.g., organizing data, reflecting on the effectiveness of actions performed, drawing conclusions)
- **communicating** (e.g., using appropriate vocabulary, communicating findings in a variety of ways)

<table>
<tr><th>Scientific Research Process and Associated Skills</th><th>Scientific Experimentation Process and Associated Skills</th><th>Engineering Design Process and Association Skills</th></tr>
<tr><td>
Communicating
Initiating and Planning
Performing and Recording
Analysing and Interpreting
Communicate results
Define the research question
Identify and select resources
Identify and record information
Analyse information and summarize findings
Scientific Research Process and Associated Skills

</td><td>
Communicating
Initiating and Planning
Performing and Recording
Analysing and Interpreting
Communicate results
Define a problem and its context
Design the experiment
Conduct the experiment
Record data
Analyse and summarize the data
Scientific Experimentation Process and Associated Skills

</td><td>
Communicating
Initiating and Planning
Performing and Recording
Analysing and Interpreting
Communicate the solution
Research and understand a problem
Ideate and generate potential solutions
Select an option and develop a prototype
Test the prototype
Evaluate and revise the prototype
Engineering Design Process and Associated Skills

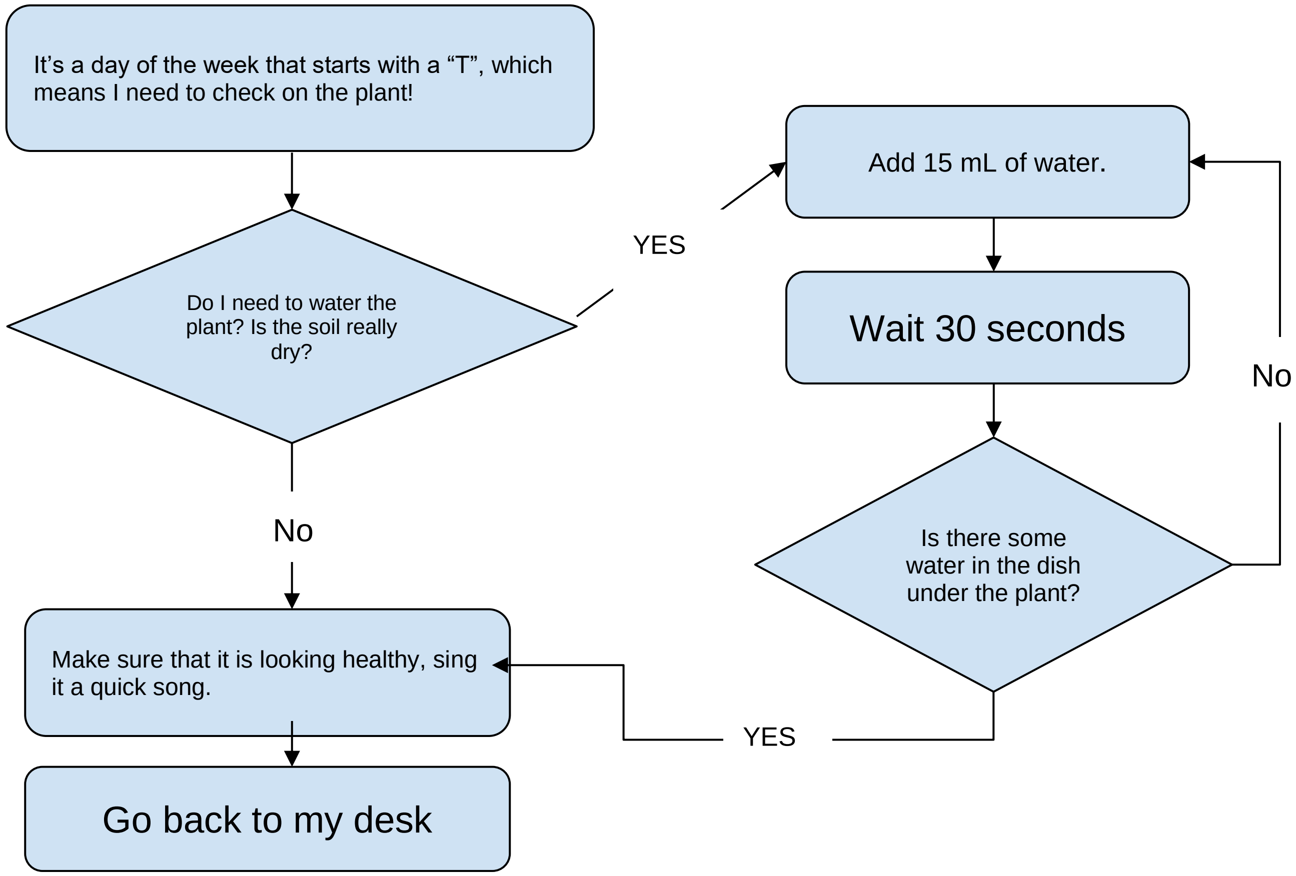
</td></tr>
<tr><th>Possible Student Activities</th><th>Possible Student Activities</th><th>Possible Student Activities</th></tr>
<tr><td>Research impact of coding and emerging technology on an industry (e.g. healthcare, transportation, retail, agriculture, etc.)</td><td>Use sensors and code to collect data to support a scientific experiment. Coding techniques could also be used to analyse and summarize the data.</td><td>Build and test a prototype of an automated system using coding and sensors.</td></tr>
</table>

Initiation and Planning

The initiation and planning (or ideation) phase of a coding project is best done in a group environment as contributions can be provided by all group members. It is important to capture the initial ideas for a new project during this phase and not worry about the implementation or coding details. When an initial set of requirements have been identified in the project initiation process there should be time allocated to **plan** the key elements of a proposed coded solution. Students should consider using **story variable** templates, **flowcharts**, and/or **pseudocode** within this phase of code planning. This provides an opportunity for student-teacher conversations prior to the implementation or coding phase of the project.

[9] https://www.dcp.edu.gov.on.ca/en/curriculum/science-technology/context/processes

Flow charts are visual representations of algorithms. Flowcharts can represent all of the fundamental coding constructions such as **sequential**, **selection**, and **repetition** tasks using different symbols.


It's a day of the week that starts with a "T", which means I need to check on the plant!
Do I need to water the plant? Is the soil really dry?
YES
Add 15 mL of water.
Wait 30 seconds
No
Is there some water in the dish under the plant?
No
Make sure that it is looking healthy, sing it a quick song.
YES
Go back to my desk

In this flowchart, the thinking behind the routine of "should I water the plant" is broken down. The diamond shapes are reserved for decisions, and usually, show the results of those decisions (for example Yes/No). Other shapes in the flowchart can be for process steps, or to collect input. In science, flowcharts can also be used to summarize the steps in a procedure.

Pseudocode is a planning and algorithm design technique using short concise steps.

```
IF (day of week is Tuesday or Thursday)
    While soil is dry (soil sensor) and no water in dish under plant (water sensor)
        Turn on water pump to deliver 15 ml of water
        Turn off water pump
        Wait 30 Seconds (recheck sensors - loop)
    Check plant for overall health
        Sing to plant a quick song
        Return to my desk
```

Performing and Recording

The engineering design process phase of **performing and recording** would involve the **writing** and **debugging** of code artifacts.

Code Debugging

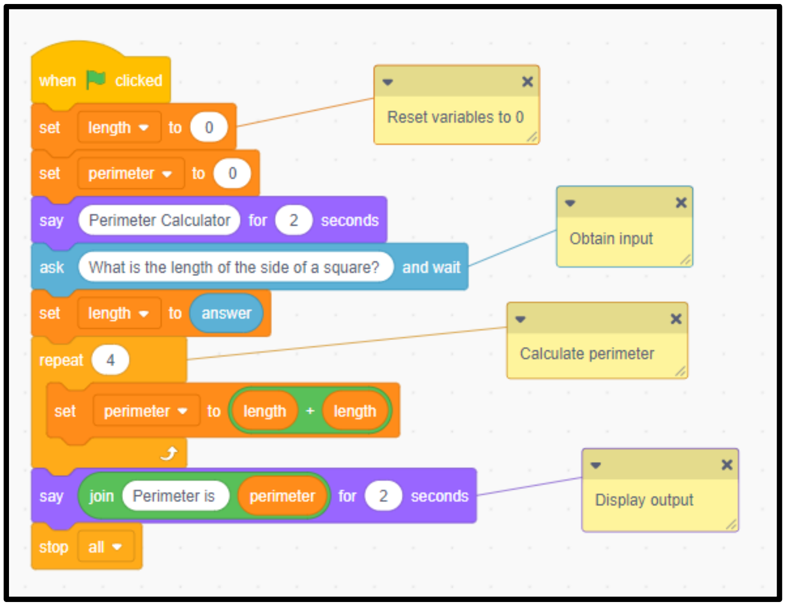
Debugging is a core skill required throughout the coding process. Debugging is the process of identifying errors and resolving the errors using various strategies. It tends to be unavoidable because often the code that is created does not work as intended.

There are some general guidelines that can assist with students' ability to debug problems. In general, the more readable the code is the easier it will be to resolve errors. For example, by using meaningful variable names that clearly represent the intended purpose then logic errors can be identified more quickly.

Debugging is an integral part of testing one’s own code. Encouraging students to code in small chunks and test frequently helps to reduce the task of debugging. If students test frequently, they have less code to sift through to find where the bug or error is, and they are also more likely to be reflecting on the function of their own work.

Code Tracing

Code tracing is a common debugging technique where a segment of code would be analyzed using a step by step approach on paper. We will explore the use of code tracing in the code example in Figure 1.



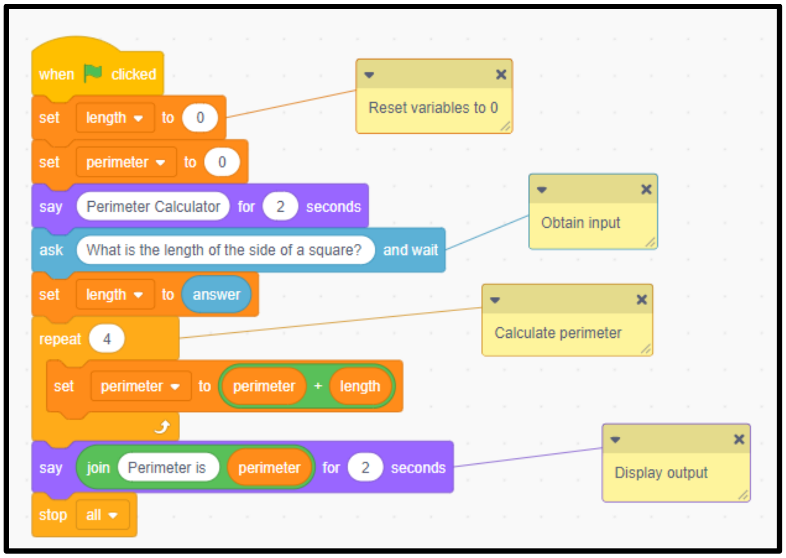
```
when ⚑ clicked
set length ▾ to 0
set perimeter ▾ to 0
say Perimeter Calculator for 2 seconds
ask What is the length of the side of a square? and wait
set length ▾ to answer
repeat 4
    set perimeter ▾ to length + length
say join Perimeter is perimeter for 2 seconds
stop all ▾
```

Figure 11: Troubleshooting logic errors in Scratch

A **code tracing table** is created by creating a table with a column for each variable and an additional column that represents the output of the program. The learner would logically step through the code and write the current value stored in each variable. If an input is obtained from a user a sample input value would be used.

length	perimeter	output
~~0~~	~~0~~	10
5	~~10~~	
	~~10~~	
	~~10~~	
	10	

Tracing is an excellent computational skill to help students understand that the **values** stored in **variables** are *temporal* and therefore any previous value stored is replaced with a new value for each assignment. The example shown here uses the set block to replace the current value of the perimeter each time. The expression block (green) is always the same and therefore we have the incorrect output for the perimeter of a square with a side length of 10 instead of the intended perimeter of 40 units.



```
when 🏴 clicked
set length ▾ to 0
set perimeter ▾ to 0
say Perimeter Calculator for 2 seconds
ask What is the length of the side of a square? and wait
set length ▾ to answer
repeat 4
    set perimeter ▾ to perimeter + length
⤴
say join Perimeter is perimeter for 2 seconds
stop all ▾
```

Figure 12: Resolved errors in a Scratch program

Collaboration

A common misconception is that computer science and coding is a solitary task that is performed by a single person in isolation. Successful coding projects involve contributions from many people with a shared common goal. There are many different perspectives that should be considered when planning, designing and implementing software solutions.

Pair programming is one technique that can be modeled for students at any age as a method of working together on a shared goal. In pair programming students perform the role of driver and/or navigator throughout the coding process.

The role of the **driver** is to use the coding language elements to represent the various elements of a potential solution while the **navigator** is tasked with providing a focus on the overall task at hand. The navigator's role is to consider alternative solutions and encourage the driver to consider possible errors that could occur. The pair work as a team using a coding project resulting in better quality results and shared ownership. The driver and navigator should be continually discussing their ideas to debug and realize their shared vision.

Assessment Strategies

Teachers can provide multiple methods for learners to express their ideas through code with the UDL framework; this is Action and Expression of coding student knowledge and skills. The Big Book of Computing Pedagogy indicates that “assessment should require students to develop code, interpret code, or both” (Big Book, pg. 59).

Coding assessment strategies should be triangulated to include observation, student-teacher conversations, and student products (Growing Success, pg 34)

Considerations for Assessment

Observations:

- **Anecdotal Observations**. As students work on their projects, educators can make observations about the strategies students are using to code, and their progress towards the learning goals and success criteria. Observations may include how students are progressing in meeting the co-created success criteria as they relate to the curriculum (i.e., the use of loops, concurrent events, etc.).

Conversations:

- **Artefact-Based Questions** (ABQs). Students answer questions about their project, the code they have written, the process, and the outcomes. By answering these questions, students reflect on their work and on what they have learned and share these insights with educators and peers.

Products:

- **Coding Portfolios / Journals.** Students can share their code in a digital portfolio with a brief explanation of what they have created. These portfolios can be housed within a Learning Management System (LMS) and can support assessment as learning and assessment of learning. An example of a coding journal prompt is provided below:

My idea is to:

Here is what I did:

Next time, this is what I’d like to fix/improve/adjust:

Single Point Rubric - Coding in Science and Technology

Next steps **Prochaines étapes**	**Meeting Expectation (Level 3)** **Répond aux attentes (Niveau 3)**	**Exceeds expectation (Level 4)** **Surpasse les attentes (Niveau 4)**
	I can design a plan before starting to code I can write, read and alter existing code I can modify my code when the outcome is not what I expected (troubleshooting) I can use the related vocabulary appropriately Grade 1. I can write clear and precise code (to tell a story) Grade 2. I can write a program that has smaller steps. Grade 3. I can test and debug my program when things do go as expected. Grade 4. I can write code that produces different outputs Grade 5. I can store numbers or words in variables to use in my code Grade 6. I can write code that takes in different input Grade 7. I can plan and design programs for a variety of purposes Grade 8. I can automate a system that has more than one part	

References

Big Book of Computing Pedagogy - Source: Raspberry Pi foundation

Grover, Shuchi. (2020). Computer Science in K-12: An A-Z Handbook on Teaching Programing

Growing Success - https://www.edu.gov.on.ca/eng/policyfunding/growsuccess.pdf

Israel, M., Lash, T. A., & Jeong, G. (2017). Utilizing the Universal Design for Learning Framework in K-12 Computer Science Education. Project TACTIC: Teaching All Computational Thinking through Inclusion and Collaboration. Retrieved from University of Illinois, Creative Technology Research Lab website: https://CTRL.education.illinois.edu/TACTICal/udl

Madkins, T. C., Howard, N. R., & Freed, N. (2020). Engaging Equity Pedagogies in Computer Science Learning Environments. Journal of Computer Science Integration, 3 (2), 1-27. https://doi.org/10.26716/jcsi.2020.03.2.1

Przybylla, M and Romeike, R. (2014) “Key Competences with Physical Computing”, KEYCIT 2014: key competencies in informatics and ICT, 7, p.351 https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/8290/file/cid07_S351-361.pdf

Sentance, S., Waite, J. and Kallia, M. (2019). Teachers' experiences of using PRIMM to teach programming in school The 50th ACM Technical Symposium on Computing Science Education: SIGCSE 2019, Minnesota.

Glossary

algorithm is a precise set of steps that describe a process.

flowchart is a visual representation of computational algorithms.

Parsons problem is a form of assessment in which learnersare asked to select from a selection of code fragments, some subset of which comprise the problem solution.

pseudocode is an informal method of designing a set of computational operations.

pair programming is a technique of developing software programs. The technique consists of a driver and navigator. The role of the driver is to edit/modify the software program and the navigator will provide guidance and assistance during the development process.

PRIMM (**Predict-Run-Investigate-Modify-Make**) is an approach to planning coding lessons and activities. The approach includes code reading skills followed by code exploration tasks. The final phases of PRIMM require students to plan and write new programs based on their interests.

syntax error is a violation (incorrect use) of the programming language.