CODING

GUIDE TO EFFECTIVE INSTRUCTION IN MATHEMATICS

TABLE OF CONTENTS

1. WHY 4

INTRODUCTION 4

DEMYSTIFYING CODING 4

"LEARN TO CODE" AND "CODE TO LEARN" 5

Pseudocode, an Indispensable Tool 5

Switching From Unplugged to Plugged Coding 5

Block-Based Coding 7

Text-Based Coding 8

DEVELOPING COMPUTATIONAL THINKING 9

The Six Mental Processes 10

Mathematical Concepts and Coding 10

Mathematical Processes and Social-Emotional Learning Skills in Coding 11

The Importance of Errors 12

Debugging 12

Coding for the Development of Transferable Skills 13

The Student's Creativity is in the Driver's Seat! 13

Communication is Essential! 13

2. HOW 15

CODING WITH INTENT 15

Integrating Coding Into Long-Range Plans 15

Planning a Coding Activity 16

High-Impact Instructional Practices in Mathematics 16

Learning Goals, Success Criteria and Descriptive Feedback 17

Direct Instruction 17

Problem-Solving Tasks and Experiences 17

Teaching About Problem Solving 17

Tools and Representations 18

Math Conversations 18

Small-Group Instruction 18

Deliberate Practice 19

Flexible Groupings 19

How to Choose a Coding Software 19

WHAT ABOUT EVALUATION? 21

Coding and Evaluation 21

Evaluating Coding and Evaluating Through Coding 21

Criteria and the Categories of Knowledge and Skills 22

How to Collect Evidence of Student Achievement 24

Observations 24

Conversations 25

Productions 27

Feedback 27

Descriptive Feedback 28

Feedback Through Questioning 28

Peer Assessment 28

3. WHAT 29

HANDS-ON 29

What Might Coding Look Like?..............................29

Unplugged Coding..............................29

Pseudocode 31

Coding in a Mathematical Context..............................32

Coding Skills 33

Progressing Through Coding Knowledge 34

Sequential Events..............................34

Concurrent Events..............................36

Repeating Events..............................38

Nested Events..............................40

Conditional Statements 43

Efficiency in the Context of Coding 46

Subprograms 47

Data Analysis Through Coding 49

Learning Situation – Grade 1 51

Learning Situation – Grade 2..............................61

Learning Situation – Grade 3..............................73

Learning Situation – Grade 4..............................88

Learning Situation – Grade 5..............................105

Learning Situation – Grade 6..............................116

Learning Situation – Grade 7 130

Learning Situation – Grade 8..............................149

4. AND THEN 159

THE STUDENT'S EXPERIENCE 159

Connections Between Coding and STEM
(Modern Problems Require Modern Solutions)..............................159

S - Science 159

T – Technology 160

E - Engineering..............................160

M – Mathematics 161

Connections Between Coding and Real Life..............................162

Connections Between Coding and Careers 164

CODING TERMINOLOGY 165

APPENDIX

MATHEMATICAL MODELLING 169

1. WHY

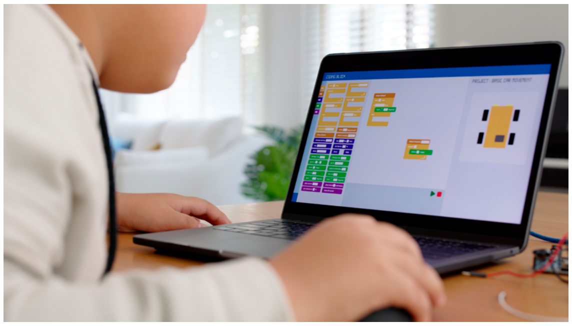
Introduction

The *Guide to Effective Instruction in Mathematics - Coding* was designed to support the classroom implementation of coding by providing effective instruction strategies and examples of successful practices. Coding is part of an even more encompassing concept: computer programming. Programmers solve everyday problems by creating code to meet various needs. To do this, they need to empathize with the people they want to help and learn about their needs. It is therefore not only about coding, but also about designing an interface to ensure that users have a pleasant experience. Finally, programming also involves testing the code with users and applying changes as needed. This guide specifically targets the stage during which the student transforms a mathematical situation into a language that can be understood by a machine.

Demystifying Coding

To better grasp what coding looks like today, it may be helpful to go back to its beginnings. While coding may be new to the classroom, it is, in fact, a concept that dates back to 1843 when a computer science pioneer, Ada Lovelace, created the programming language for coding. It was in the 1950s that several languages appeared, such as Assembler, COBOL and LISP. These languages could seem very daunting to novices, so that at the time only computer scientists had the privilege of coding computers.

It was in the 1980s that the first "mainstream" programming language finally appeared. People were able to code a budget, an interactive cookbook or even a video game. Since then, coding and programming languages have evolved greatly, making them accessible to everyone, even young children. In the early 2000s, graphic programming using "blocks" appeared, reducing the incidence of syntax errors found in text-based languages. Several interfaces, robots and microcontrollers using block-based coding have appeared.



Coding helps get computers and devices to perform tasks. In order for students to see the relevance, it is important to make them aware that coding and math are all around us. Whether it's the smart thermostat, the stove, the remote control, the connected watch, the electric toothbrush, the autonomous car, these devices all contain code and all use mathematical concepts to function. For example, the toothbrush uses angle measurement and force measurement to gather data on brushing habits, the thermostat and stove use comparisons (>, <, or =) to check if a certain temperature has been reached, and the GPS in a smartphone uses distances and trigonometry to predict the time required to get from point A to point B. Teachers are encouraged to take advantage of all everyday contexts to make connections.

Coding is accessible to everyone. Assistive interfaces and technologies now help students with special needs to code. Deafness, blindness, limited mobility or cognitive challenges, for example, no longer present a barrier to learning coding. Several accessible coding platforms are now available online and many items are available from special education providers.

"Learn to Code" and "Code to Learn"

While it is initially important for the student to learn to code by acquiring the rudiments of code blocks or language syntax, it is crucial to understand that the purpose of coding in the classroom is to "code to learn." With this in mind, students use coding to explore new concepts in mathematics or consolidate existing learning.

Exploring new concepts: the student could create code that allows thousands of draws per second to compare experimental probability with theoretical probability. The student could also attempt to draw polygons to determine regularities in the exterior and interior angles based on the number of sides. In this case, the student creates simulations using the code to draw conclusions.

Consolidating existing learning: the student could use coding to demonstrate understanding of the concept of relative location of objects (for example, having a character move left, right, above, below an object) or use the Pythagorean theorem to code a move into a right triangle shape. In these cases, the student already knows the mathematical concept and uses coding to represent it, allowing them to apply it and consolidate their understanding.

While many activities and lesson plans exist online for telling stories or creating video games using coding, it is important to remember that the main focus here is on representing mathematical concepts. These activities can help with learning to code, but for the purposes of assessing expectation, ensure that the student is also coding to deepen their understanding of mathematical concepts. The concepts and situations to be represented can, and should, come from all strands. Coding can be an important support to spiral planning, as students can always build on their knowledge and experiences to represent new situations and solve new problems. Therefore, coding should not be taught only once at one time.

Pseudocode, an Indispensable Tool

The term **pseudocode** simply means that the sequence of instructions the computer will have to follow is written in a familiar language and organized as a sequence of code. It is much less intimidating for the student to speak a language that is familiar to them before they have to use code blocks or a computer language like Python or JavaScript. Also, pseudocode doesn't belong in any language, so the organization and syntax used must be understood by the student and their peers, not a computer.

Pseudocode is also an interesting approach to debugging, which is the detection and correction of errors in code. By writing pseudocode and comparing it with non-functional code, the student can find the place where an error has been inserted.

Examples of pseudocode in this guide are often organized in tabular form, and can be found in the learning situations and in the section on pseudocode.

Switching From Unplugged to Plugged Coding

When we code without a computer, it is **unplugged coding**, a name that refers to the fact that we do not need to be "plugged in" to the computer to code. When the student completes a mathematical task that resembles a code sequence, it is called unplugged coding. **Pseudocode** is also a particular form of unplugged coding and can act as an effective intermediary to move from unplugged to plugged coding.

As students begin to learn to code, teachers often use unplugged activities to introduce the concept. These activities help the student gain a very concrete understanding of concepts like sequencing without the use of a computer, tablet, robot, or microcontroller. For example, in an unplugged activity, one person might take the role of the coder and another might take the role of the machine. One person might give a sequence of instructions to a peer to shoot a basketball hoop or give a sequence of instructions for two other students to clap their hands.

The student will realize that, even if his or her instructions are rudimentary, the student acting like a machine will often fill in the missing information. For example, if one says "raise your hand" without giving a distance, the student will not continue to raise his or her hand indefinitely. The use of unplugged coding in a learning situation is discussed in the Hands-On section.

Although unplugged activities can serve as a warm-up for coding, it is also important to switch to the machine so that the student experiences its limits, which will not complete the missing information.

The term **plugged coding** is used to describe coding that is done on the computer. There is a huge variety of software and languages with which code could be written, as we mentioned in the introduction. That said, we often hear about "block-based coding," which is creating visual code using blocks with different functions, and "text-based coding," which is using characters to write our code. Characters can be words, abbreviations or symbols that are recognized by the programming language in question.

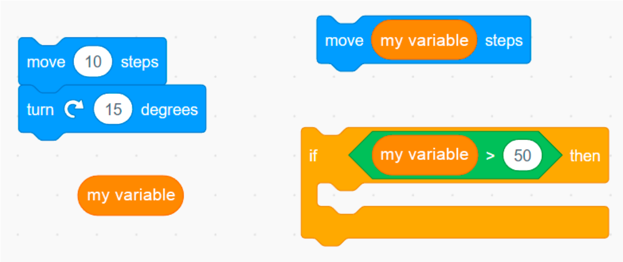
It is important to move from **unplugged** to **plugged** coding to show the difference between following a procedure in a human being and in a robot. For example, a human can use prior knowledge to make predictions and complete imprecise instructions, whereas a computer does not have this ability. The instructions will be followed to the letter. This can sometimes be a source of frustration for students. Comments such as "My robot didn't do what I asked!" are often heard. The advantage of using code to solve mathematical problems is the speed with which the computer can represent and solve complex problems; for example, in a probability experiment, the more data collected, the closer the experimental probability approaches the theoretical probability. The computer will be able to generate the result of a random outcome thousands of times before a human being generates a dozen. This ability opens the door to the modelling of complex situations and to simulations that are not possible with unplugged coding.

There is no specific time to move from unplugged to plugged coding. In fact, it is possible to move from one to the other in an interleaved fashion throughout the school year. The important thing is to recognize students' comfort with coding in mathematical contexts. Assigning tasks that help them further explore coding software in a deliberate, purposeful, and well-timed manner is essential. If, during a coding activity, it is observed that students are having difficulty representing a given mathematical situation with code, there is nothing to prevent going back to write pseudocode as an unplugged activity followed by converting the pseudocode to a programming language to validate and verify the algorithms involved in problem solving.

Block-Based Coding

In the context of coding and programming, block-based coding is a very recent idea if we compare its creation in the 2000s with that of the first coding language in 1843! Block-based coding makes coding accessible to a larger population despite the fact that we often use it as an introduction with our students. Many educators use block-based coding platforms with text-based coding language to generate proofs of concept quickly before moving on to text-based coding.

Block-based coding offers several benefits to the student starting their coding learning journey. In a **block-based graphical language**, the student can assemble sequences of instructions using blocks that each have a function. The student does not have to type long sentences in a language that seems unfamiliar. Each block looks like a puzzle piece that fits together. In order to facilitate student understanding, the blocks take on specific shapes depending on what they represent; for example, a numeric value could take a rounded shape, while a boolean value (logical value that corresponds to true or false) would take the shape of a hexagon.



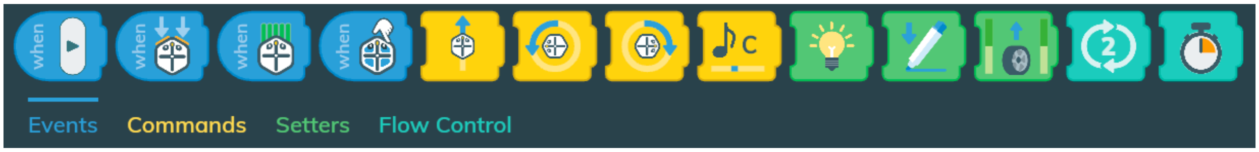
Here we see some examples of blocks that are compatible or not with each other.

Puzzle-shaped blocks can be put together to make a sequence, but it is not possible to attach an oval-shaped block to the sequence, because oval-shaped blocks represent quantities in this software. It would therefore not be logical to give without context two directions followed by a quantity. However, it is possible to add an oval block inside a block that contains a value (in an oval).

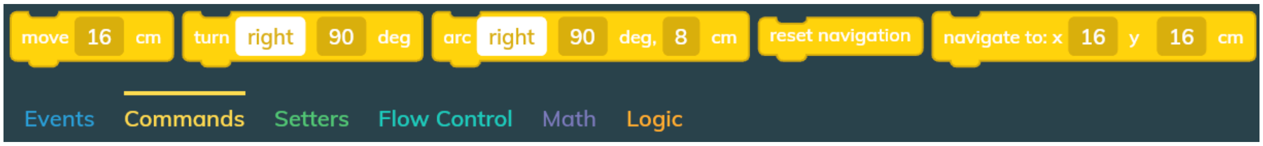
Similarly, it is not possible to add a quantity (oval) by itself in a block that requires a Boolean value (hexagon), because the Boolean value must correspond to a question that is answered with yes or no. If we write the sentence dictated by the code, we can see this logic. The instruction "If (my variable) is greater than 50, then... " makes sense.

During coding, if the student fails to attach one block to another, it is because they are not compatible. This particularity of block-based coding allows a considerable reduction in syntax errors, that is errors related to the structure of the code itself. Moreover, when there is text associated with the blocks, the text is in common language and not in programming language. Some block-based coding software even offer the possibility of converting code into a text-based coding language to support the student in the transfer of block-based coding into languages like Python and JavaScript.

In order to meet the needs of students, several languages offer two different versions: one with icons (requiring little or no reading) and one with descriptions.



Example of block-based coding software that uses icons

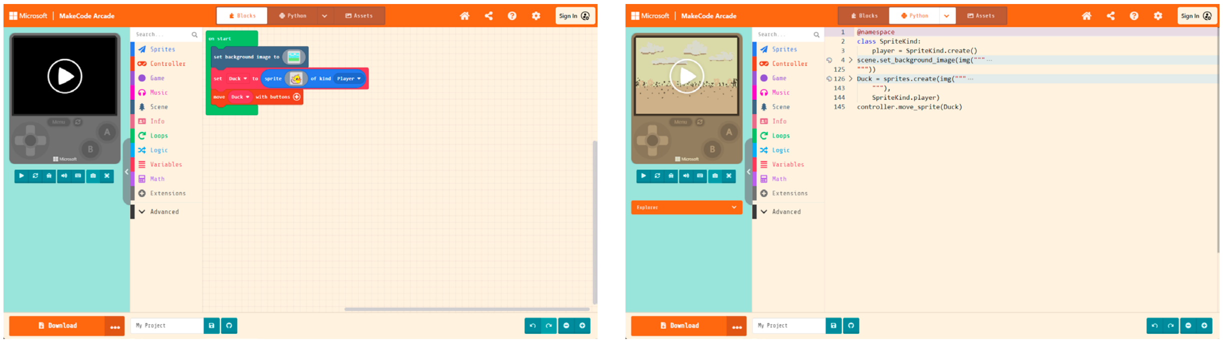


```
move 16 cm
turn right 90 deg
arc right 90 deg, 8 cm
reset navigation
navigate to: x 16 y 16 cm
```

Example of block-based coding software that uses text

Text-Based Coding

Text-based coding can be seen as the "next step" and is what is used in industry. While this type of coding may seem intimidating at first, there are several tools that make learning easier. The student who is able to make the transition from block-based coding to text-based coding can use coding platforms that help make the conversion with one click. The student does not start from scratch. It is possible to switch between modes and make changes. As of the writing of this guide, the popular languages are Python and JavaScript. As the field of computer science evolves and new problems to solve are discovered, text-based coding languages are also evolving. That being said, the student will quickly discover that all languages are similar and use similar structures: sequences, loops, conditional structures, variables, etc. It should also be noted that, in the vast majority of cases, text-based coding languages are based mainly on English terminology. Comments, on the other hand, can be written in any language. Comments are discussed in the Communication is Essential! section.



```
on start
    set background image to
    set Duck to sprite of kind Player
    move Duck with buttons
```

```
@namespace
class SpriteKind:
    player = SpriteKind.create()
scene.set_background_image(img("""...
"""))
Duck = sprites.create(img("""...
    """),
    SpriteKind.player)
controller.move_sprite(Duck)
```

Microsoft's MakeCode coding software allows students to switch between blocks and Python at the click of a button. This helps the student see, in real time, the text equivalents of their code blocks. Many block-based coding software programs offer this feature.

Developing Computational Thinking

To teach a machine to perform a task, the student must "translate" his or her thought or intent into a language that can be understood by a computer chip. This conversion process is possible through **computational thinking** (also called *algorithmic thinking*). Just as students are provided with the tools to develop their algebraic thinking, proportional reasoning, and probabilistic thinking, it is crucial to have strategies in place that help the student increase their computational thinking by providing opportunities to practice. Students need to practice "thinking like the machine".

It is essential for the student to be able to recognize that computer chips process information very differently from the human brain. The human brain can handle instructions that are blurry by making inferences. The machine, on the other hand, needs very precise and exact instructions, including numerical values.

For example, if a person is taught to raise a bottle of water to their mouth and tilt it to drink water, extremely precise instructions would not be necessary.

- Raise the water bottle.
- Bring to mouth.
- Tilt until water runs into mouth.
- Return to upright position when mouth is full.

The human being can use inference, gesture and modelling to make another human being understand the task. Teaching this task to a robot is much more difficult. For example:

- Tilt the glass to an angle of 0˚.
- Move the glass along the *y* axis over a distance of 20 cm.
- Move the glass along the *x* axis over a distance of 8 cm.
- Start rotating the motor clockwise to tilt the glass up to a 120˚angle.
- Wait 5 seconds.
- Start rotating the motor counterclockwise to return the glass to a 0˚ angle.
- Move the glass along the *x* axis over a distance of - 8 cm.
- Move the glass along the *y* axis over a distance of - 20 cm.

This example shows how precise and detailed writing instructions for a machine must be. Several concepts of measurement, geometry, and number sense are also used. Depending on the intent and context, the student could make a connection to algebra by using variables to represent distances, time, and angles so that the code can be applied to different situations (for example, someone who is taller, someone who drinks faster).

The Six Mental Processes

Computational thinking is developed through **six mental processes**.

MENTAL PROCESS	EXPLANATION
Decomposition	A problem or task may seem difficult to solve. Decomposition involves recognizing the different subtasks in the problem and breaking down the task. This allows the student to solve the problem "in bite-sized pieces".
Pattern Recognition	In a problem, certain actions or elements may occur more than once. The student must therefore recognize them and use coding structures, such as repetition, or recycle the code.
Use of **Algorithms**	Problem solving is done using algorithms; for example, the student may recognize that in order to solve a problem, a sorting algorithm, a comparison algorithm, or a moving algorithm is needed. In designing these algorithms, the student designs a step-by-step list of steps to solve the problem. **Note:** Teachers can make the connection to the algorithms used to learn operations.
Abstraction	Some problems include superfluous elements that do not serve to solve the main problem. The student must therefore abstract the essence of the problem by dropping the insignificant details. These details can be added after resolving the problem.
Modelling and **Simulation**	The model must be created by converting the algorithm into a language understood by the device, then execute and simulate its operation. The student can visualize how their code works.
Evaluation	Assess whether the solution addresses the problem and determine whether to alter the code.

Note: It is important to note that these are processes, not steps. There is no specific order in which to complete them.

Mathematical Concepts and Coding

The student will quickly realize that mathematical concepts come to life and prove to be very useful. This is a perfect opportunity to answer the age-old question, "Why are we learning this? For example, a student who moves a robot by indicating the number of rotations of the wheels may ask, "What happens if four rotations of the wheel are not enough, but five rotations of the wheel are too many? This provides an ideal context for introducing decimal numbers. The exploration could continue with the question, "What happens if 4.5 turns of the wheel are not enough, but 4.6 turns of the wheel are too many? This is now an ideal context for making the transition from tenths to hundredths position values. This example provides a sense of the quantity that a tenth and a hundredth represent in a real and relevant context. The student could also use proportional reasoning, "I think my object should move at least two and a half times as far." The student is constantly in a problem-solving process.

Mathematical Processes and Social-Emotional Learning Skills in Coding

> There is strong evidence that developing social-emotional learning skills at school contributes to students' overall health and well-being and successful academic performance. It also supports positive mental health, as well as students' ability to learn, build resilience, and thrive[1].

Like any other aspect of teaching, the integration of coding into the classroom requires commitment from both the teacher and the student in order to be successful. When we talk about commitment in the school context, we are talking about the motivation to learn, the desire to participate, to know the instructional intent behind the various activities and to recognize and believe in our abilities. How then can we create an environment conducive to learning when we are in such a vast and, for many, unknown field?

In the beginning, it is important to create a safe space and develop trusting relationships with students. The limitless possibilities of exploration using coding are both a gift and a possible source of frustration. It is important to recognize the emotions that the student and the educator might feel in a situation where the task won't be successful the first time. These emotions become opportunities to develop social-emotional skills in order to develop a positive identity as learners of coding. An intentional conversation about the emotions felt during the task (pride, confusion, stress, discomfort, doubt, enthusiasm) allows recognition and validation of the realities of the class, including those of the teacher, who is also placed in a situation of new learning. Modelling the recognition and management of emotions through an accessible model can make the practice easier for the student to understand and perform. It's also a great opportunity to create a sense of community. Coding is learned in a group, and you don't need to have prior expertise in order to take advantage of coding as a tool in mathematical contexts.

For example, the student who says "I don't know how to do it" or "it's too difficult" may be experiencing cognitive overload because of their high emotional state, which makes learning difficult, if not impossible. Teachers can start by acknowledging their emotions and encouraging them to take a step back. When the student shows an openness to learning coding, teachers can ask questions about their prior knowledge in order to find a similar context that will serve to reassure the student about their ability to succeed or offer them avenues for exploration that could yield similar results. The goal is to place the student in a situation of success.

Coding can also be used to develop skills such as perseverance. During a coding activity, the student does not need to ask the teacher to validate their code. The student, seeing the outcome following the execution of the code, immediately knows if the code gives the expected result. Subsequently, the student may decide to make changes, either by altering a functional code to make it more efficient or more precise, or by "debugging" the code which does not give the desired outcome. The process is very rich and very rewarding, helping the student not only to experience the mathematics that surrounds them, but also to develop their socio-emotional skills in a caring and safe learning context.

It is important to make the student aware that, although teaching machines tasks may seem tedious, machines have important advantages. In assembly lines, they can repeat tasks precisely without burning out. Computers can also make millions of calculations per second without making mistakes (if they've been programmed properly!).

[1] The Ontario Curriculum. Mathematics, Grades 1-8 Ontario Ministry of Education, 2020.

The Importance of Errors

The importance of errors is based on neuroscience. There is research that has proven that it is when humans make mistakes that the brain "lights up", where we see regions of the brain that are active[2]. In some cases, we even see that the regions of the brain responsible for memory and attention are only active as a result of errors. It is in these precise moments, when there is error, that the learning is at its peak, that the student is more apt to recall the concepts learned and to transfer the learning to a new context.

Therefore, we must not forget to value mistakes in coding. Fortunately, by its very nature, coding is best learned by making mistakes. There is little or no risk when the student experiments in a coding sequence - the worst thing that could happen is that the code doesn't work. Therefore, it is important to approach coding in a mathematical context with an artistic perspective, where creativity and risk taking can be appreciated and valued. Each error experienced by the student is a new learning experience. Even if the outcome is not what was intended, it is often possible to discover how certain elements of code interact by making mistakes. Sometimes exploring possible solutions and eliminating errors is more important than knowing all the answers.

Error can easily cause stress in students. Teachers can model situations where they do not have the correct answer to the question asked. Solving a problem together with code can show how far we can go without necessarily having all the coding knowledge. Students see teachers editing the code, making suggestions, searching the Internet, and even taking breaks if frustrations mount. At the end of the process, when the desired outcome is achieved, students will have experienced modelling the processes of coding. Remember that even computer programmers rarely get it right the first time!

Debugging

Sometimes the problem-solving process during coding can become frustrating for students, especially when there is a bug in the code that seems untraceable. The term *bug* is used in computer science to refer to an error that interferes with the proper functioning of the code. When the student looks for bugs to fix them, the term "**debugging**" is used. In order to help the student develop social-emotional skills when difficulties are encountered during debugging, several strategies can be used:

- Close the code and forget about it for a few minutes or hours. Upon return, the student will see elements not perceived during the initial coding.
- Eliminate all non-essential lines of code to focus on the main algorithm.
- Ask another person to look at the code for a second opinion.
- Use the **pseudocode** before starting to write the code and refer to it throughout the coding process.
- Execute the code one line at a time to determine when the bug occurs. Some coding interfaces offer the "Execution Tracking" option. The student can also verbalize the intent of each block to find inconsistencies.
- Include a sound file or on-screen message at some point in the code to check if that part of the code is reached during execution.

[2] Science of Learning Research Center.

Coding for the Development of Transferable Skills

Coding can also be a gateway to the development of transferable skills. Learning to code and learning through coding develop several skills that are not exclusive to coding. The use of coding in mathematical contexts can have a positive impact on the student in all subject areas. The following two examples focus on the skills of innovation, creativity and entrepreneurship, as well as communication and collaboration, but all seven transferable skills are present in the examples and learning situations throughout the guide.

The Student's Creativity is in the Driver's Seat!

Although several tasks presented to the students include a procedure with specific parameters, the coding tasks always help them to let their creativity run free by altering the code, adding new functionalities to it, perfecting the user interface, improving the user experience and taking care of the visual appearance.

For example, a student working on a currency conversion algorithm could add flags to make the program more appealing, fetch real-time exchange rates, allow the user to drag and drop for ease of use, add instructions and sound effects, add administrative transaction fees if needed, and add text-to-speech to improve accessibility. Although the students in this example were given the same instruction (to create a currency conversion algorithm), the projects will be very diverse and will demonstrate limitless creativity. As a teacher, this creativity should not be curtailed by limiting students who think outside the box, but rather celebrated.

Coding is also a sandbox for risk taking. The student should be encouraged to alter the code to observe the outcomes. Starting with existing code, even if it seems unintelligible at first, the student can make small incremental changes to it and see how the outcomes vary (for example, changing the value of a variable, changing the threshold value of a comparison in a conditional structure, changing the number of repetitions in a loop). This process of investigation will help them deepen their understanding of the code and the programming language used.

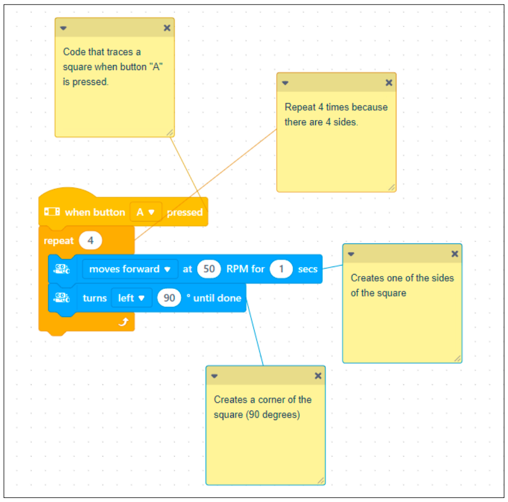
An important benefit of taking risks in coding is the absence of negative repercussions. It is not possible to damage the device, the robot or the microcontroller by changing the code. The worst that can happen is that the code is no longer functional. It is always possible to go back! The student should therefore be encouraged to dissect and rework their code.

Communication is Essential!

Communication is an essential skill in all subjects and it is especially so in coding. In industry, several programmers often work on the same project. When code gets very long, it can be difficult to know what each line of code does, even for experts in the field.

All computer languages provide the possibility to include comments. These comments are ignored by the device during code execution, but they allow humans to better understand its structure. In the case of block-based coding, comments often take the form of removable sticky notes. In a textual language, a character is used to tell the device to ignore what follows; for example, in Python, the # symbol is used to indicate a comment.

This documentation of code is not only helpful to the student, who has a better understanding of their code when it is reviewed, but also provides an opportunity for teachers to collect evidence of learning for assessment purposes.



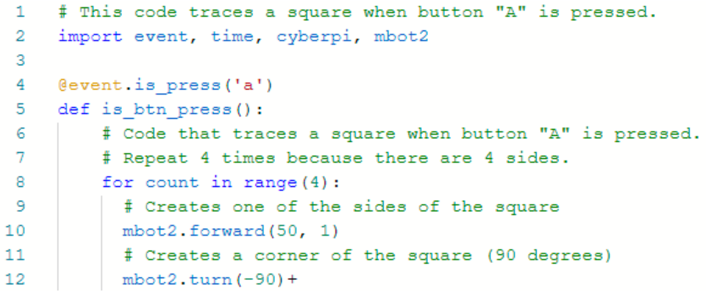
when button A pressed
repeat 4
moves forward at 50 RPM for 1 secs
turns left 90 ° until done

Code that traces a square when button "A" is pressed.

Repeat 4 times because there are 4 sides.

Creates one of the sides of the square

Creates a corner of the square (90 degrees)



```
# This code traces a square when button "A" is pressed.
import event, time, cyberpi, mbot2

@event.is_press('a')
def is_btn_press():
    # Code that traces a square when button "A" is pressed.
    # Repeat 4 times because there are 4 sides.
    for count in range(4):
      # Creates one of the sides of the square
      mbot2.forward(50, 1)
      # Creates a corner of the square (90 degrees)
      mbot2.turn(-90)+
```

2. HOW

Coding With Intent

Integrating Coding Into Long-Range Plans

While coding can be integrated into a variety of subjects, such as literacy, art, science, and technology, it is essential that the integration be done for the purpose of exploring, learning, or consolidating mathematical concepts.

The form that coding will take in long-term planning depends on several factors. Fortunately, coding can be inserted at any point in a long-range plan.

For example, unplugged coding, in which the student plays the role of the computer, is an excellent entry point for explaining computational thinking to students. As a bridge to computer coding, a consolidation task related to a mathematical concept could be presented using block-based coding software. In this way, the student can really focus on exploring the software, since the mathematical concept would already be learned.

With practice, students' comfort with representing mathematical situations with coding increases, and increasingly complex mathematical situations can be represented with code. Since coding is a gateway that is accessible to students and has a very high ceiling, giving students a single opportunity in the form of a coding "unit" is a great disadvantage. It is important to take advantage of the versatility of coding for the purpose of diverse learning. The student may even choose to use coding to represent and solve a mathematical problem without being required to do so by teachers, much like the student choosing to use manipulatives.

The many ways coding can be integrated into the mathematics course make coding very useful in a spiral planning model. In the spiral approach, there are three ways to use coding.

1. The same mathematical concept is represented multiple times throughout the year, using increasingly complex programs; for example, a series of movements can be presented in unplugged coding and then transformed into pseudocode, a sequence of blocks, or textual code. It would also be possible to use this approach to present specific coding concepts, such as repeating events in a sequence, then in a loop, and finally with variables to represent the values in the code.
2. Coding software is used to represent a mathematical concept that becomes increasingly complex; for example, the student can use conditional statements (if, then, else) to collect user input (which becomes data), analyze it in list form, and calculate measures of central tendency.
3. Coding knowledge and skills and mathematical situations become increasingly complex over the course of the year; for example, using properties of the rectangle to write a code, and then changing the parameters of the code to create a variety of geometric shapes. Gradually, the code could also be used to locate, in a Cartesian plane, the vertices of the geometric shape using precise coordinates, and then to make transformations.

Fortunately, coding software and languages allow for the saving of completed work. It is therefore possible for the student to consult or alter a code throughout the year without having to redo it completely each time. (Note that in order to be able to save, some coding platforms require the creation of an account. It is important to ensure that the software or platform respects the data management guidelines of your school board.)

Planning a Coding Activity

Like any educational activity, the success of a coding activity depends on its organization and intent. Thus, the planning of the coding activity should not be neglected. This planning can contain different steps, such as the choice of the coding platform or the learning and exploration time of the chosen platform in order to help the student develop knowledge and confidence. Including students in the development of an activity is paramount to making the targeted math relevant and interesting. This creates a closer and lasting connection between the student and the mathematics around them, develops their sense of representation, and therefore their sense of accountability.

The starting point for planning a coding activity should be the instructional intent related to the mathematics curriculum. It is important to clearly target the instructional intent so that the coding really serves to represent a mathematical situation.

Learning goals are central to the planning of an activity and flow directly from the instructional intent. The question is: What will the student be able to do at the end of the activity? By explaining these learning goals to students, teachers give meaning to the activity by guiding their questioning with leads. What is the intent of this statement? Students should stay on task? The learning goals can then guide the student to the targeted learning.

In addition to learning coding, it is important to provide students with opportunities to learn **with** coding. In the context of a math class, this means that a mathematical concept or situation is represented with code, giving the student a new perspective on mathematics. It is also possible to have the student read code that has already been sequenced to predict the outcome using their prior mathematical knowledge.

The structure of a coding activity is not different from an activity in any other area or subject. There are the same three stages: warm-up, exploration and consolidation of learning. It is important to remember, however, that there are targeted coding knowledge and skills for each grade, and that **coding will need to be presented in conjunction with another mathematical concept**. A coding activity may therefore have learning goals and assessment criteria related to the knowledge and skills of coding as well as to another mathematical concept, situation or strand that is being modelled with coding.

Throughout the process, it is important for teachers to collect evidence of learning by observing and listening to student exchanges, in addition to collecting final products. Coding assessment will be discussed later in the guide.

High-Impact Instructional Practices in Mathematics

Several ways exist to leverage high-impact instructional practices in the coding context. The use of high-impact instructional practices should also be intentional and planned. The following is an overview of the use of high-impact instructional practices in situations in which coding is used to introduce, practice, or consolidate a mathematical concept. The list is not exhaustive, however, and teachers should select those practices that best align with the instructional intent.

Learning Goals, Success Criteria and Descriptive Feedback

Coding, by its very nature, is a vast and complex subject. This complexity has several advantages, but two major disadvantages:

- the student may not know where to start;
- the student is distracted by the many possibilities and moves away from the intention.

Both of these disadvantages are offset by clear learning goals, which help students know precisely what elements of a coding platform are under study at any given time.

> As essential steps in assessment for learning and as learning, teachers need to share learning goals and success criteria with students at the outset of learning to ensure that students and teachers have a common and shared understanding of these goals and criteria as learning progresses[3].

Since coding will be used as a tool for representing mathematics, the criteria will need to be specific, explain the targeted coding skill, and indicate how it applies to the mathematical context under study, where relevant.

For example, instead of saying "I can use a loop to repeat a sequence of code", we can make the criterion even more precise and targeted by saying "I can represent a multiplication using loops".

Note that descriptive feedback is further explained in the evaluation section of the guide.

Direct Instruction

Direct instruction can pass information to a group of students or check for understanding one student at a time. Student responses become evidence of learning. Teachers can provide immediate feedback.

Problem-Solving Tasks and Experiences

Often known as a "three-step lesson," "three-act math lesson," or "collaborative classroom," these are similar to the learning situations presented in this guide. Initially, students' prior knowledge is activated and then they are invited to choose an effective strategy to demonstrate their understanding of the concepts. In coding, it is interesting to see the devices or software that the student chooses to do the work. Of course, coding tools must be seen and explored for the student to be able to choose one that they like and that meets the requirements of the task.

Teaching About Problem Solving

It is very rare, even for professional coders, to create a perfectly functional code sequence on the first try. Creating code requires knowledge of the functions of blocks or commands, but knowing how they interact requires trial and error. By trying several combinations of blocks or commands, the student creates a bank of possibilities to use in a future project or problem solving.

[3] Growing Success, p. 28.

In addition to trial and error to develop coding problem-solving skills, it is also necessary to practice reading sequences of code to find the error. This process, also known as debugging, requires an understanding of how blocks or commands interact and invites students to take risks and make mistakes in the search for the "bug." Students must analyze and understand the purpose of the code and the role of the sequences within it. Each debugging process creates new knowledge that can be mobilized during a future debugging process.

Tools and Representations

Coding can be a tool for representing a mathematical situation. In addition, as with manipulatives, there are many ways to represent the same mathematical situation with code. Teachers need to target times when the topic of study is conducive to the use of code in order to put students in a position to succeed.

Tools and representations are not answers to mathematical problems, but facilitators of questioning and thinking. Using the code as a tool, the student will need to think about many things and make decisions. As there is a progression in their coding skills, the student will be able to use more complex sequences and specialized blocks to make their code unique and representative of their thinking.

Math Conversations

It is possible to have math conversations with students as they explore a concept. This will help teachers see what students are learning, how they are progressing, and how comfortable they are with the various tools.

Possible questions to start conversations:

- What do you notice?
- What are the similarities between the lines of code? What are the differences?
- How would you like to change the line(s) of code?
- What code blocks did you use to arrive at the solution?

All of these open-ended questions encourage discussion about coding and the mathematics represented by coding. This conversation could be part of a warm-up before launching a coding activity or serve as a consolidation.

Small-Group Instruction

The accessible, yet high ceiling nature of coding makes coding tasks very suitable for small-group instruction. While a small group is receiving targeted instruction, the productivity of the rest of the class can continue. This is where the variety of approaches to solving a problem with coding can be advantageous. Students not receiving direct instruction from teachers can still work toward learning goals through their own methods. Learning occurs during this type of free experimentation. During small-group instruction, the learning goals will need to be fragmented so that sessions are short in duration, but long enough to allow for exploration.

Deliberate Practice

In order to improve, you need to practice. Understanding the basic concepts of coding is important. Learning takes place during experimentation with the blocks or controls available to the student.

When students are less comfortable with coding, teachers' approach may resemble modelling to make students' code work. Then, during experimentation, students can alter, move, delete, and add blocks or commands to discover what possibilities exist with the software or coding language in question. This experimentation promotes the learning of concepts. Teachers are available to support students and correct erroneous approaches.

As students become more comfortable with coding, the initial modelling can be shortened or eliminated. This allows students to independently experience a mathematical situation. This is now deliberate practice, as students develop comfort with coding by applying concepts learned during modelling and experimentation.

Flexible Groupings

This approach gives students the opportunity to work in the grouping that best meets the instructional intent of the activity to allow them to learn more effectively. This flexibility lends itself well to coding, and is even a method often used in the programming industry.

During a coding activity, groupings could differ according to intent or progression; for example, teachers could model the use of a code to the class. After this initial exploration, students work individually to complete the assigned challenge. Teams of two are strategically created by the teacher so that students can check their code or ask questions. Students then form small groups or discuss as a class to present their solutions.

This task could also be done with random groupings that would give students the opportunity to work with new people and expose them to different ideas and ways of thinking. Heterogeneous groupings open the door to targeted instruction for students' particular needs. These groupings help teachers move more freely around the classroom to observe, assist and check for understanding.

Practices such as flexible groupings help students discuss and present their learning and discoveries throughout the activity. Teachers can also adapt their instruction and strategies based on the grouping of students.

How to Choose a Coding Software

In order to make learning through coding accessible to all, it is important to choose coding software that students can use independently. Visual block-based coding software (for example, Scratch^{Jr}, Hopscotch, code.org) or robots with movement-only functions (for example, Bee-Bot) would be ideal to start with.

As students develop oral communication skills as well as reading skills, a wider range of software and platforms become available for block-based coding (for example, Scratch, MakeCode) and robotics (for example, Ozobot, Sphero, Cue). Platforms that feature a tutorial (for example, Blockly, code.org) could give students the opportunity to explore coding software at their own pace before tackling more complex mathematical problem representations.

Block-based coding is a great way to start representing more complex problems by giving students a little more freedom and choice in their coding, and limits the number and type of errors using shapes and colour lines of code. Many robotics platforms and microcontrollers (for example, Micro:bit, Arduino, Adafruit) have a block-based coding structure.

For students who can make the leap to text-based coding, there is excellent intermediate software between block-based coding and text-based languages (for example, MakeCode, Swift Playgrounds). These software programs can make the transition easier by converting block-based code to JavaScript or Python code.

At the intermediate level, block-based coding software is an excellent tool for representing moves and operations in a very visual way, but may not be the best tool for some strands, such as Financial Literacy. A spreadsheet (for example, Excel, Google Sheets) would be a better tool for performing budget-related calculations, although it is possible to do so with block-based coding software. Data collection can also be done with block-based coding software, but a dedicated platform for data collection (for example, Google Forms, Mentimeter, Poll Everywhere) can make data analysis much more efficient. In block-based coding software, for example, calculating a median in a list of data requires more than 10 lines of code, whereas in a spreadsheet, it takes just one.

The world of coding is constantly evolving. It is important to regularly monitor technology in order to discover software or platforms that are adapted to the needs of the students and the targeted instructional intentions.

Before choosing a software, platform or coding device, it is important to think about the instructional intent of its use and the current level of the students. There are a few questions to ask first.

- Does the software, platform or device require special approval before purchase or use?
    - Does the software, platform or device require the creation of student accounts?
    - Has the software, platform or device been approved by the school board?
- What is the instructional intent of using coding?
    - Is specialized software or equipment needed to represent mathematical situations?
    - What activities can be accomplished with this software or device?
- What is students' current level of proficiency in coding?
    - Is this the first time coding has been addressed, or do students already have a foundation to build on for new knowledge?
    - Can students move on to text-based coding?
- What is the profile of the class? Are the task and software chosen accessible?
- Is there a cost associated with the chosen software, platform or device?
    - Will this expense be helpful to students in other grades?
- Will the software, platform or device allow the student to make choices?
    - Will it be possible to offer differentiated options with the chosen tool?

What About Evaluation?

Coding and Evaluation

> Computational thinking helps nurture problem-solving skills, logic, and creativity. And technology is transforming every industry on the planet. Students today should learn how to create technology, not just use it[4].

When planning an assessment, it is important to keep in mind the ultimate goal of the assessment: to improve student learning. While it is possible to assess learning based on the final product of a coding activity, a significant amount of information related to student learning can be gathered throughout the process. Observations taken during the coding activity as well as conversations overheard between students are all evidence of learning. What does not change is the importance of clear, specific, and relevant assessment criteria and student engagement in their creation to give them a sense of ownership and accountability for the work at hand[5].

Coding with blocks or text requires a lot of trial and error, traces that are difficult to document. However, this context allows teachers to provide students with feedback that is descriptive and immediate, yet positive as well as constructive. The important thing is that the feedback focuses on a specific coding knowledge or skill and is linked to the assessment criteria. This feedback helps the student adjust and focus on next steps (which can also be done in teams with peers or teachers). The very nature of coding encourages teachers to use visual cues to know when and how to provide guided instruction to students who need it (for example, by strategically creating work groups or providing direct instruction of a concept to the entire class).

In assessment as learning, students become "masters" of their learning with autonomy by setting personal goals based on learning goals and feedback from teachers and peers. The accessible, yet high ceiling nature of coding encourages students to set personal challenges or goals based on their interests while staying true to the learning goals and assessment criteria. The coding task then becomes much more than creating a series of instructions. The task becomes an entire learning experience that allows teachers to gather evidence and students to grow and explore topics and problems that interest them[6].

Evaluating Coding and Evaluating Through Coding

The coding expectation is assessed when students engage in coding activities. Students read, write, alter, and predict the outcome of their programs to demonstrate their coding skills and knowledge. It is important to remember that coding, in the context of the mathematics classroom, must be used to represent and solve mathematical situations. Coding becomes relevant when it is linked to another mathematical concept, as coding can be difficult to assess on its own.

When coding is paired with another mathematical concept, it becomes a form of representation, much like manipulatives, to help the student better understand mathematics; for example, the creation of code having repeating elements, or loops, could be done when learning repeated addition or repeating patterns. The student can then use the code in an evaluation context in order to show their understanding of the effect of repetition on the mathematical situation.

Assessment with coding requires the student to use coding to demonstrate skills in other mathematical strands. This allows for the simultaneous assessment of both coding skills and mathematical understanding. This will certainly affect the rubric discussed below.

[4] Hour of code.

[5] Growing success, p. 38.

[6] Growing success, p. 31.

Criteria and the Categories of Knowledge and Skills

Since teachers assess coding skills as well as skills in other mathematical concepts, it is important that the achievement chart be clear and specific so that the student knows the expectations for each mathematical strand if more than one is being assessed.

There are three possible scenarios regarding coding assessment.

In the first, only the coding is evaluated. In this case, the linked mathematical concepts are not assessed and are therefore not part of the achievement chart. The achievement chart should contain criteria that relate only to coding knowledge and skills.

In the second, both the coding and the mathematical concept are assessed. The achievement chart must then specify whether the criterion is specific to coding, specific to the mathematical concept or specific to both.

In the last, coding is not assessed. Students may choose to represent a mathematical situation using their knowledge and coding skills. The achievement chart focuses instead on the mathematical concept under study.

Note that in all three scenarios, coding is always used in a mathematical context.

The following are examples of skills that make coding assessment in mathematics possible, written in the form of success criteria. Some criteria explicitly require that they can be assessed alongside a mathematical concept if that meets the instructional intent. It is also important to note that the co-construction of success criteria remains an approach to be prioritized and that classroom criteria may differ from these.

KNOWLEDGE AND UNDERSTANDING

- I **write** and I **execute** lines of code related to mathematical situations.
- I **indicate** the blocks (or commands) and the actions initiated by each block (or command) related to the targeted mathematical situations.
- I **understand** that I **represent** a series of instructions using an algorithm.

THINKING

Use of Planning Skills

- I use **pseudocode** to **plan** the structure of my code.
- I can **break down** my code into sections with specific outcomes and **make changes** to meet a specific intent.

Use of Processing Skills

- I can **debug** non-functional code sent by the teacher or my peers.
- I can **spot logic** errors (the code works, but doesn't produce the expected outcome) and **syntax** errors (the code doesn't work at all).

Use of Critical and Creative Thinking Processes

- I can **check** the structure and functionality of my code.
- I can constructively **critique** a code to determine different ways to achieve the same outcome.

COMMUNICATION

Expression and Organization of Ideas and Understandings

- I **express** and **organize** ideas and information in a way that makes it easier to read code or pseudocode.
- I **respect** the syntax of the programming language under study.
- When a code is altered, I can **describe** the impact of the changes on the outcomes.

Communication for Different Audiences and Purposes and in Oral and Written Forms

- I use **notes**, **comments** or **multimedia elements** (sounds, videos) in my code to facilitate its interpretation and review.

Use of Conventions and Terminology in Oral and Written Forms

- I use **terminology** associated with coding knowledge and skills (for example, sequential events, concurrent events, repeating events, nested events, control structure, operator blocks, conditional statements) to express my ideas.

APPLICATION

Application of Knowledge and Skills in Familiar Contexts

- I can **read** a code, then **alter** it in order to adapt it to a precise mathematical situation.

Transfer of Knowledge and Skills to New Contexts

- I can write code that represents a precise mathematical situation **using** familiar language, pseudocode or another programming language.

Making Connections Within and Between Various Contexts

- I can point out **real-life** math problems or those related to **my interests** that could be solved by coding.
- I can use code to represent **a new** mathematical situation.

These examples of criteria are not suitable for all situations. It is possible that in the early stages of learning a new programming language (textual or block-based) the focus may be on following syntax and recognizing blocks or commands and their functions. As the student progresses in learning coding, the focus of the assessment should also progress.

Coding also makes it possible to assess multiple strands at the same time. For example, consider an activity in which the student must write a code to draw a rectangle and determine its area and perimeter. In this activity, we find skills in coding (Algebra), measurement (Spatial Sense) and operations (Number). Coding then becomes a powerful representation tool that makes other mathematical concepts possible[7].

How to Collect Evidence of Student Achievement

When evaluating coding, the entire process can be evaluated, from the start of a blank screen to the final product. Teachers can collect evidence of learning through triangulation related to assessment criteria. The student should have several opportunities to receive feedback and adjust accordingly. Teachers need to collect evidence of learning from multiple sources and at multiple times throughout the learning sequence. The three vertices of the triangulation—observations, conversations, and products—can all be assessed in different ways. Here are some examples.

Observations

Throughout the coding process, students need to apply their knowledge and make new ones. It is important that the teacher be present throughout the process in order to take into account the application of prior knowledge and the application or transfer of new knowledge and skills. Additionally, the strategies employed to explore and debug code are excellent sources of evidence that the student is actively learning. The teacher can use tools such as an observation grid, photos or videos taken during the design of the project. The observation grid must target knowledge inspired by the success criteria. With a grid, it is possible to record observations quickly, efficiently and continuously.

Examples of Observation Grids

An initial observation grid could be colour-coded to determine the degree of achievement of the criteria in question. The grid could be done on the computer or on paper using highlighters to facilitate observations. In the following example, green indicates that a criterion is met, yellow indicates that the student is on track, and red indicates that the student will need to continue to practice in order to meet the criterion. The comments will serve as inspiration for the descriptive feedback that will be given to the student regarding the knowledge and skills observed.

The criteria for the task could be the following:

- I can plan a sequence of code using diagrams or pseudocode.
- I correctly use a variety of blocks in my code in order to achieve a specific outcome.
- I can explain the reason for the differences between my pseudocode and my current code.

[7] Translated from CFORP training *Le codage, oui, mais comment l'évaluer?*

NAME OF THE STUDENT	PLANNING MY CODE	APPLYING MY CODE	JUSTIFYING MY CODE	COMMENT
	🟩	🟨	🟥	

A second evaluation grid focuses on specific and observable knowledge and skills related to the established success criteria. Teachers use symbols to record observations; for example, the circled letter indicates the skill has been acquired, the crossed out letter indicates that the skill is being acquired, and a cross (X) over the letter indicates that the skill has not been acquired.

The success criteria for the task could be the following:

- I alter my lines of code to correct syntax or logic errors.
- I am creating lines of code to collect data and put it into a list.

We could use the letters D (debugging), W (writing lines of code) and L (list data).

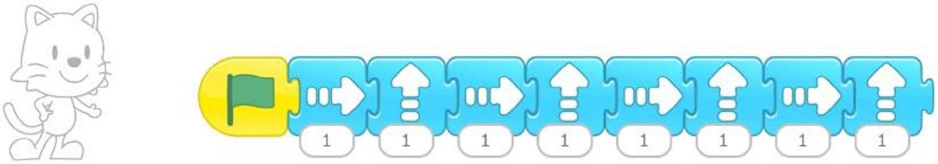
Name of the student ↓	Date →	
		D W L

As teachers circulate among students, they will be tasked with observing the degree to which students apply the debugging process (for example, breaking code into smaller sequences to find an error more easily), use a variety of blocks to make their code more efficient (for example, creating loops for repeating events), and use comments or a logbook to note the differences between their pseudocode and their actual code, much like a rough draft before publishing a text.

Conversations

In the course of creating a code, conversation comes to play a large role. Since students will often use strategies such as trial and error or peer feedback, it is important to have conversations related to the issues. These conversations are often prompted by questioning.

Example



- Can you explain to me what the pixelated or digital image, also known as *sprite,* will do?
- Why did you choose to put the blocks in this order? Could you find another way to move your *sprite* to the same place?
- I notice that your *sprite* takes a step to the right, a step up, a step to the right, a step up... Is there a way to reproduce this same movement?

Conversations can be between students, between teacher and student, or between teacher and a small group of students. It is important to make time in the schedule to converse with students or listen to students in conversation. Through conversation, students explain and extend mathematical concepts or reasoning.

Questions to Ask

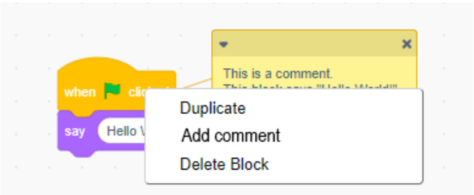
- What words would you use to explain what you had to do in this task?
- What could you do different?
- What approach did you use?
- How do you know?

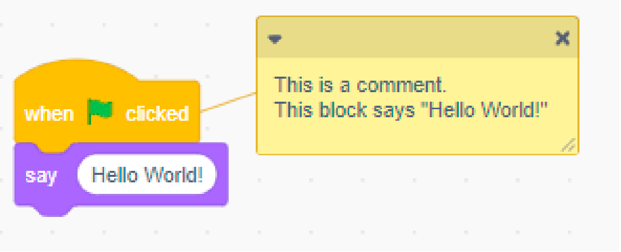
Coding is a team sport, both in the classroom and in the professional programming world. It is important to show your code to colleagues (or, in this case, to other students in the class) in order to receive feedback and constructive suggestions. Active listening among students will help to recognize feedback that shows a good understanding of concepts. That being said, once a student is able to communicate comfortably in writing, comments in code are also a great way to gather evidence of learning, as they are conversations between the person who wrote the code and the person who will read it. Comments, in a code, can be done in a variety of ways.

Here are two common examples of a block-based coding software (Scratch) and a text-based programming language (Python).

Block-Based Coding:

A right click on the block we want to comment will display a tab in which there is the option **Add comment**, which looks like a sticky note.





This is a comment.
This block says "Hello World!"

Python:

The number sign (#) followed by a space indicates the beginning of a comment. Each line of the comment must begin with these two characters. Comments are ignored when the code is compiled.

```
# The next line of code prints "Hello, World!"
print("Hello, world!")
```

Products

It is often implied that production is the end product, but this is not always the case. A production, in the context of coding, can also be the pseudocode written before a coding task begins, or a logbook that explains the challenges encountered while coding and the debugging strategies that were applied. This type of evidence of learning helps assess the coding process in addition to the final product.

Production helps the student apply prior and acquired knowledge, and demonstrate mastery of learning goals. The assessment of the final product should focus on the code and how the code meets the success criteria. It is sometimes easy to lose sight of the success criteria when a code seems very complex or exceeds the desired level of complexity. In this case, it is important to return to the criteria in order to offer a fair and objective assessment.

With the self-assessment linked to the success criteria in addition to their code, the student essentially gets a checklist to help them revise and readjust their code according to the criteria. All code, including comments, should be delivered as production, whether using a share-by-link feature, a screenshot, or, in the case of text-based code, a transcription of the code in a word processing software.

Finally, a key element in the assessment of the production is the choice of the tool in relation to the success criteria. The choice of tools to create a code must be diverse and accessible so that the student chooses the tool that can create a product that highlights the targeted outcomes. It is also important to differentiate the type of production for the same group of students. By achieving the learning goals with descriptive pseudocode, creating a sequence in block-based coding software, or programming a robot, the student has more choices and, therefore, is able to find the approach or tool that best meets their needs. The differentiation of productions also helps to value differences, and encourages students to learn from their peers by observing the productions of others.

Feedback

> When there are multiple opportunities for feedback and follow-up, all students gain skills in assessing their own learning as they reflect on the success criteria[8].

Feedback is an integral part of learning. For feedback to be successful, it must be helpful, accurate and caring. The dialogue with the students must be present, on several occasions, in the learning situation. The student must be allowed to adjust and make changes, if necessary, as a result of feedback from peers or teachers. As in all other subjects, feedback in coding should be based on the success criteria and should be done in collaboration with the student. Keep in mind that the student participates in the feedback.

[8] Source: High-Impact Instructional Practices in Mathematics.

Descriptive Feedback

Feedback, whether from teachers or peers, is essential in coding. Feedback helps students not only to secure their learning, but also to feel confident with the various tools. Descriptive feedback throughout the process helps students achieve targeted goals while solidifying their learning.

Some precision is required in the information provided to students for it to be descriptive feedback. For example:

INSTEAD OF...	TRY
Good work!	You have used a wide variety of blocks in your code!
Your code is very complex.	You created a complex sequence of blocks to achieve your goal. Do you think you can reach your goal with fewer blocks?
You haven't quite succeeded in the task. Try again!	Could you review the order of the commands in your code so that they follow the syntax of the programming language correctly?

These are just generic examples. In the classroom, teachers should provide feedback that relates to the achievement of the success criteria.

Feedback Through Questioning

Questioning and descriptive feedback are not mutually exclusive. Rather, questioning can be the first step toward even more descriptive and focused feedback by engaging the student in the feedback process. This allows the student to think about the purpose and method rather than simply creating code and waiting for the next steps. This approach also has the advantage of allowing teachers to confirm the achievement of targeted learning goals.

The following are some examples of generic questions that could lead to a conversation and a very focused feedback. It is the teacher's job to tailor the questioning to the targeted success criteria.

- Could you show me how you created this code?
- I see that you used the "variable" blocks. What effect did these blocks have on your code?
- Can I suggest you something? I see that you have added the "movement" block several times. Do you know a block that allows you to repeat a movement several times? Show me how you can alter your code.
- Could you show me the code blocks checklist and their function? I would like to check something together.

Peer Assessment

It is not only teachers who can provide feedback. On the contrary, the feedback at the time of a peer assessment can be even more powerful than that of the teacher. Peer assessment is not a tool to influence a student's final grade on the achievement chart, but rather is an opportunity for students to showcase their accomplishments, ask each other questions, and offer constructive criticism or suggestions. Peer assessment should therefore not be restricted to the final product, but rather should occur multiple times throughout the coding process.

Modelling this practice at the beginning of the year is essential to its proper use, and students need opportunities to practice playing this role within the class. Students are able to develop their skills in order to give feedback that helps their peers, for example, to create useful and effective code. This feedback also gives them the vocabulary and experience to evaluate themselves more effectively. Peer assessment is also an opportunity to provide evidence of learning in the form of conversations.

3. WHAT

Hands-On

What Might Coding Look Like?

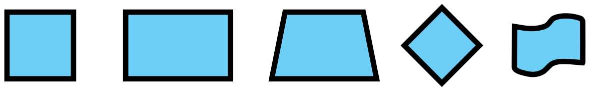
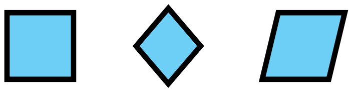
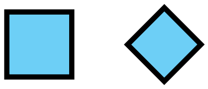
Many still think that coding is represented by computer screens covered with incomprehensible text and symbols, sometimes even resembling a foreign language! It's a very common image, but coding can take many forms, and some elements of coding are even easier to tackle without a computer!

The next sections discuss a possible progression in the use of different forms of coding.

Unplugged Coding

As mentioned in Part 1, **unplugged coding**, or computer-free coding, is a very powerful tool that helps students better understand how a computer processes information. This type of coding is very much like a walkthrough - a sequential list of directions, beginning with imperative or infinitive verbs, with a clear purpose. First, students can be asked to "think like a computer" - that is, to follow the walkthrough word for word, without making assumptions and without using prior knowledge.

For example, by asking students to draw a square, it is possible for them to access their prior knowledge, remember the properties of the square, and then draw it on paper. However, if students are asked to describe to a person how to draw a square, the task becomes interesting - especially if the person, who is acting as the "computer" in this activity, follows the directions exactly without making assumptions!

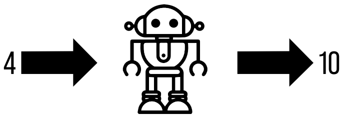
EXAMPLE OF AN INSTRUCTION GIVEN BY THE STUDENT (INPUT)	EXAMPLES OF DRAWINGS OF THE PERSON (OUTPUTS)
"Draw a shape with 4 sides."	
"Draw a shape with 4 equal sides."	
"Draw a shape with 4 equal sides and 4 right angles."	

In this example, the number of "OUTPUTS" possibilities decreases with the complexity of the directions. During an unplugged coding activity such as this, it is possible to observe and listen to the conversations to gather evidence of the student's conceptual understanding. A student who has simply memorized the shape of the square or who has not fully grasped the associated mathematical vocabulary will have difficulty providing clear instructions.

Unplugged coding can also be used to introduce or expand knowledge in other strands. The algebraic robot, for example, helps students imagine the "code" that transforms one number into another number.

For example:

Students provide the numbers 4, 9 and 3. The teacher, who knows the "code" being sought, writes this:



After the first transformation of the robot, the students try to find the possible lines of code to give such an answer.

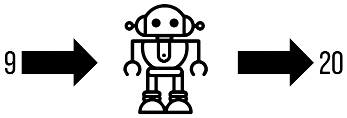
Examples of possible codes:

"The robot adds 6 to the number."

"The robot multiplies the number by 2, then adds 2 to the answer."

"The robot multiplies the number by 2.5."

Then the robot can transform the next number.



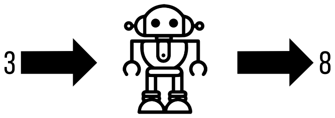
Students can check lines of codes to see which ones always give a correct answer, or even create new lines of codes if needed.

Examples:

"The code that adds 6 to the starting number doesn't work, because 9 + 6 doesn't equal 20."

"The robot multiplies the number by 2, then adds 2 to the answer."

Then it is possible to move on to the next number.



Conclusion example:

"The code where the robot multiplies the number by 2 and adds 2 to the answer works for all three cases."

If there is no consensus on the code, students again provide inputs for the robot to transform until there is consensus on the secret 'code'. In addition to providing an opportunity to work with equations and inequalities, this activity helps students see that it is possible to have more than one correct answer when writing code, and that trial and error is an important part of coding.

Unplugged activities can be enriching and rewarding on their own, but in order to explore the power of coding in a mathematical context, one must move from unplugged coding to plugged coding. Unplugged activities are a prerequisite for computer coding.

Pseudocode

Pseudocode can act as an effective intermediary between unplugged and plugged coding. Since there is no connection to a specific language or programming software, writing pseudocode helps the student describe the actions to be performed by the computer and the desired outcome. The student can use familiar language and terminology to create a plan for the code. This stage is similar to the draft in the writing process - the big ideas remain stable, but there will be many changes between the pseudocode and the final code.

For example, a group of students are working on the concepts of area and perimeter, and want to create code that will allow a *sprite* to move along a rectangular path, and then say the area and perimeter of the rectangle. Here is what the pseudocode would look like:

```
Sprite
    position = lower left corner
    orientation = to the right
Variables
    [base] = base of the rectangle
    [height] = height of the rectangle
    [area] = area of the rectangle
    [perimeter] = perimeter of the rectangle
Starting condition = true
    Request: "What is the base of the rectangle? "
        Value of [base] = response
    Ask: "What is the height of the rectangle?"
        Value of [height] = answer
        Value of [area] = [base] * [height]
        Value of [perimeter] =
        [base] + [height] + [base] + [height]
    Repeat 2x
        Move - [base] step forward
        Rotation - ¼ turn clockwise
        Move - [height] step forward
        Rotation - ¼ turn clockwise
    Say "The area of the rectangle is [area]."
    Say "The perimeter of the rectangle is [perimeter]."
End
```

The location of the text in the pseudocode helps us follow the sequence of instructions; for example, instructions to repeat under the statement "repeat 2x" are more indented than the statement itself. Students develop their own ways of writing pseudocode with their own abbreviations and symbols. This flexibility makes coding more accessible to students as they learn to use coding software. Pseudocode can also serve as evidence of learning, showing the student's understanding of the mathematical situation to be coded and of computational thinking, even if the student is not yet proficient with coding software.

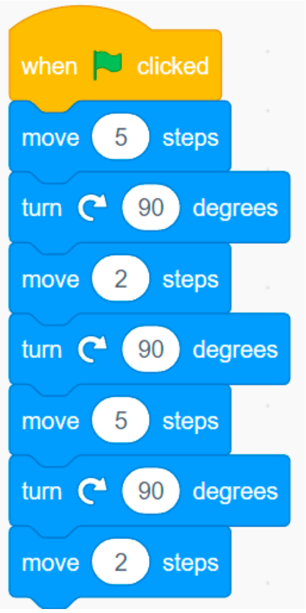
Coding in a Mathematical Context

A clear and precise mathematical intention should be the starting point of a coding adventure. Effective coding instruction focuses on **solving problems** and **representing mathematical situations computationally**. These two components are the basis for the use of coding throughout the primary, junior and intermediate divisions. The skills and related concepts will, for their part, provide a progression of learning.

It is essential that students learn to use code in mathematical contexts in order to meet this expectation. However, it must be remembered that the use of coding software itself does not necessarily meet this expectation if the code written, executed or altered is not directly related to a mathematical context.

For example, coding software is often used to animate a character, commonly called a *sprite*, using directional commands. The following pseudocode and block-based code could show the movement of a cat on a rectangular track.

```
Sprite "cat"
        position = center
        orientation = right
Starting condition = true
        Move - 5 steps forward
        Rotation - $\frac{1}{4}$ turn clockwise
        Move - 2 steps forward
        Rotation - $\frac{1}{4}$ turn clockwise
        Move - 5 steps forward
        Rotation - $\frac{1}{4}$ turn clockwise
        Move - 2 steps forward
```



```
when 🏁 clicked
move 5 steps
turn ↻ 90 degrees
move 2 steps
turn ↻ 90 degrees
move 5 steps
turn ↻ 90 degrees
move 2 steps
```

The student who creates this pseudocode, or similar code in coding software, is certainly demonstrating coding skills, but the context in which they were applied is not necessarily one of problem-solving or mathematical representation. In order to make this situation relevant to the mathematics curriculum, one must first consult other strands to determine how coding can make the learning or application of a mathematical concept more relevant and accessible. Coding becomes primarily a tool for representation and mathematical problem-solving, and cannot be taught and assessed alone.

Coding Skills

Coding skills can be summarized in two broad themes: **writing** code and **reading** it. There is a similarity to effective literacy instructions, in that learning to write and read is most effectively done in an intertwined manner[9]. Thus, we do not have to teach an entire language or software code before we can take advantage of the learning opportunities initiated by coding; for example, the student could be given a piece of code in order to observe the outcome of its execution, and then **alter** certain places to customize it. Alternatively, the student could receive code that contains errors and attempt to **predict the outcome** of its execution and try to correct it (otherwise known as *debugging)*. In both of these examples, students are **reading and writing code** in an intertwined fashion, without necessarily needing a comprehensive knowledge of all the terms and features of a software program or coding language.

Solving problems with code requires the activation of prior knowledge on the part of the student, as well as a significant amount of resourcefulness and curiosity. By its very nature, coding lends itself well to trial and error - error being a very powerful learning tool, as it is rare to be able to write code that solves a problem on the first try. In order to solve the problem, students must first build on their prior knowledge, try different strategies, make connections and draw conclusions.

Coding is also a **tool of mathematical representation**. It is difficult to give tasks that allows students to practice and develop the skills found in the curriculum content without contextualizing the tasks with other mathematics strands. Coding can and should primarily serve as a representational tool – much like manipulatives – to help students better understand a mathematical situation. The examples provided in this guide are therefore only starting points.

Writing code, the act of writing textual instructions or assembling code blocks, is not the end of the creation process. In order to verify that the code is working as expected, it must also be **executed**. Executing the code may involve a single step, such as clicking a flag or an "execute" button, or it may involve compiling the information found in the code into a set of instructions that can be understood by the computer. In the case of a robot or microcontroller, for example, the code must be uploaded to the device before it can be executed. Note that single-step execution also includes a "compilation" step, but this is behind the scenes, giving the impression of instantaneous execution.

[9] For more details, see A Guide to Effective Literacy Instruction.

Progressing Through Coding Knowledge

There is a sequence to teaching coding knowledge to ensure that students have some basic knowledge before moving on to more complex code. The suggested sequence is as follows:

GRADE	CODING KNOWLEDGE TO BE PRIORITIZED
Grade 1	Sequential events
Grade 2	Concurrent events (+ sequential)
Grade 3	Repeating events (+ sequential, concurrent)
Grade 4	Nested events (+ sequential, concurrent, repeating)
Grade 5	Conditional statements (+ sequential, concurrent, repeating, nested)
Grade 6	Code efficiency, with all control structures seen in grades 1-5
Grade 7	Subprograms and defined count, with all control structures seen in grades 1-6
Grade 8	Data analysis for informed decision making, with all control structures seen in grades 1-7

Although this sequence of teaching knowledge related to coding is logical, with knowledge that allows access to more and more complex functions from year to year, it is not a limit. Students certainly discover features in the languages and coding software used that go beyond the knowledge to be prioritized in that grade. This curiosity and this desire to create more complex lines of code must be allowed, encouraged and valued, because it is a way of engaging students in mathematics.

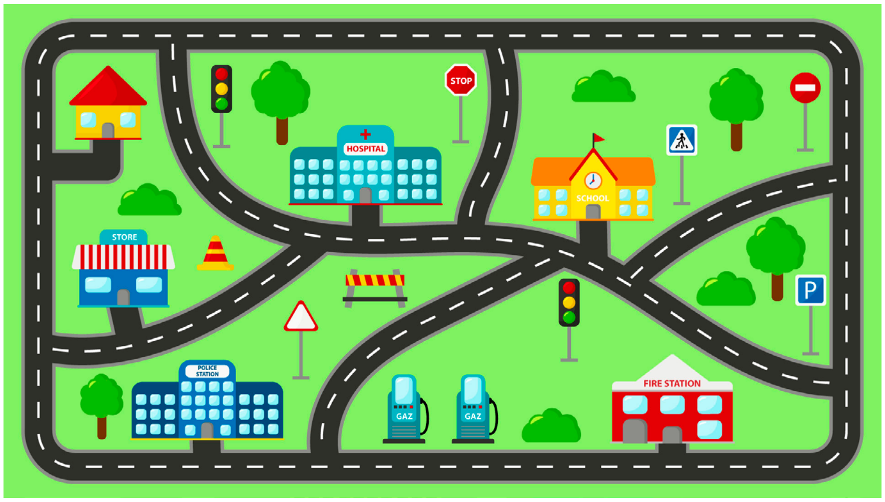
Sequential Events

The sequence is really the basic structure of coding. The term *code sequence* is also often used to discuss coding. It is important to start with the sequence and types of actions that can be done.

In block-based coding software, the sequence in which the blocks are placed determines the order in which the instructions will be executed. In this context, coding essentially becomes a walkthrough. The types of events in a sequence are not limited to translations. Sounds, visual changes, and even moments of waiting can be part of a sequence and become sequential events.

Sequential events: set of instructions executed one after the other, usually from top to bottom or from left to right on a screen.

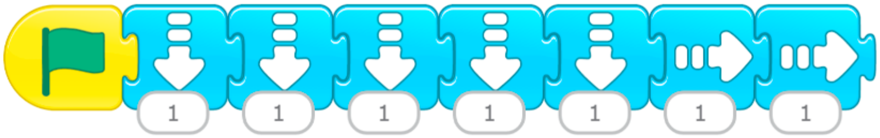
Let's take the map of a fictional city.



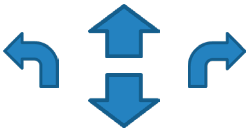
Visual coding or robotics software (for example, block-based coding) would allow the student to animate a *sprite* so that it could move to specific locations. A pseudocode to perform a translation from the house to the police station, following the paths, could look like this:

```
Initial position = to the left of the yellow house
Starting Condition = true
        Move 5 steps
            Direction = down
        Move 2 steps
            Direction = right
Final position = in front of the police station
```

The code corresponding to this pseudocode, as programmed in block-based coding software, might look like this:



This is just one example of a movement sequence that would get the character to their destination. It should also be noted that, in certain coding contexts, the possibilities of movement are more limited; for example, some robots frequently used in the classroom only use the following commands:



In this case, the vocabulary *quarter turn, clockwise* and *counterclockwise* replaces left and right. Directives, on the other hand, remain sequential events, as they are executed one after the other.

Still according to the map of the fictitious city, here is a possible pseudocode in order to leave from the police station to go to the school, always following the paths:

```
Position = Police station
Starting condition = true
        Move 6 steps
            Direction = right →
        Move 2 steps
            Direction = up ↑
        Move 4 steps
            Direction = diagonal ↗
```

In most cases, block-based coding software does not have the ability to give diagonal movement instructions, which would be necessary in order to meet the goal and respect the pseudocode. It is necessary to use horizontal and vertical movement **concurrently** in order to have this effect.

Concurrent Events

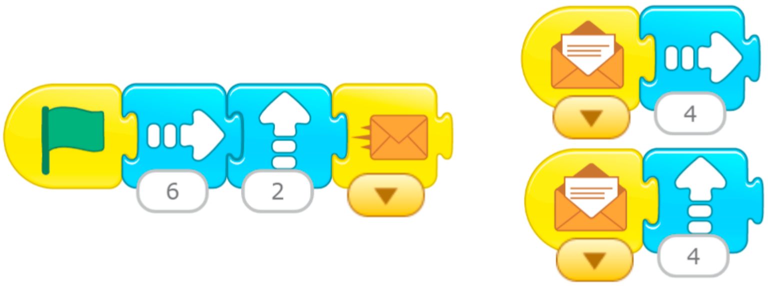
The meaning of the word *concurrent* is well known. Note, however, that concurrent events in the context of coding are events that are triggered by the same action. Just think of concurrent events, like a car race. The green light signals the start of the race and all cars start the race at the same time. The cars won't necessarily perform the same maneuvers or drive at the same speed, but they all received the same start signal. These are **concurrent** events, even if the actions (speeds, manoeuvre, etc.) are not identical.

Concurrent events: several events occurring at the same time.

A pseudocode that allows the *sprite* to go from the police station to the school therefore becomes:

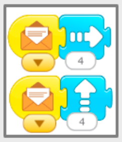
```
Position = Police station
Starting Condition = true
        Move 6 steps
            Direction = right →
        Move 2 steps
            Direction = up ↑
Begin Diagonal Movement
        Move 4 steps
            Direction = right →
            Direction = up ↑
```

The sequence in block-based coding software might look like this:



In order to better visualize the code, the student could also overlay his or her code on his or her pseudocode. This helps to better identify errors, if any, and then correct them, while leaving evidence of the student's understanding of the mathematical concepts being studied. This practice can be very useful at all levels, especially when learning to use a new software program or a new coding language.

Position = Police station
Starting Condition = true
Move **6** steps Direction = **right** →
Move **2** steps Direction = **up** ↑
Begin Diagonal Movement

Move **4** steps
Direction = **right** →
Direction = **up** ↑ 

Note: The coding software used in this example employs the "send message" and "receive message" events. When this code is executed, arrival at the "send message" block (closed envelope) simultaneously triggers all sequences that begin with the "receive message" block (open envelope). Coding software can vary in imaging to achieve this same outcome.

The student could also use two different *sprites* that navigate the map at the same time, which also satisfies the requirement for concurrent events.

Repeating Events

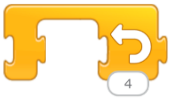
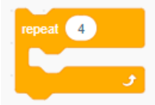
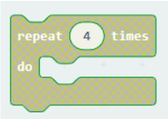
Sometimes the very nature of a mathematical situation requires repetition. For example, repeated addition is used to improve conceptual understanding of multiplication, and repeating pattern is a prerequisite for growing patterns and pattern recognition.

In the coding world, the term *loop* is often used to name a repetition. This structure also makes it possible to simplify the code by limiting the number of blocks or lines necessary to achieve a desired outcome.

Repeating events: events that repeat themselves. In coding activities, loops are used in code to repeat instructions.

In block-based coding software, the loop often takes the form of a square parenthesis with the code to repeat inside. Here are some examples of different block-based coding platforms:

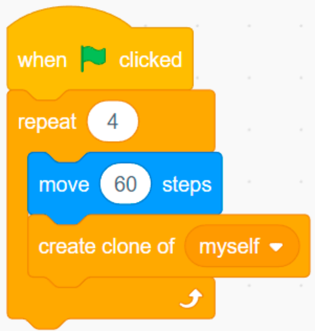
SCRATCH J^{R}	SCRATCH	MAKE:CODE


4
	
	repeat 4
		
		repeat 4 times
		do

In each case shown above, the code inside the loop would be repeated 4 times.

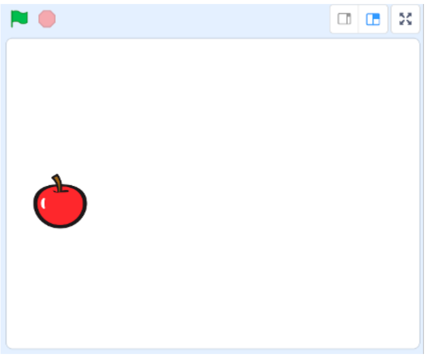
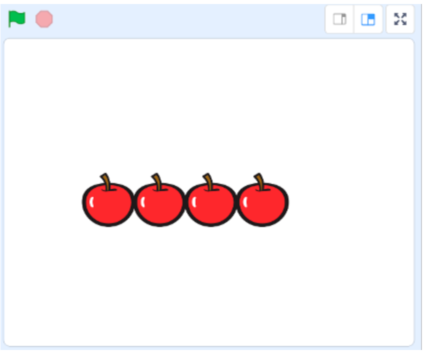
Since the loop is analogous to multiplication due to its nature, the use of repeating events is ideal for solving problems that require multiplicative thinking.

We can then use the loop to represent a multiplication. Take the following code:



```
when 🏳 clicked
repeat 4
    move 60 steps
    create clone of myself ▾
```

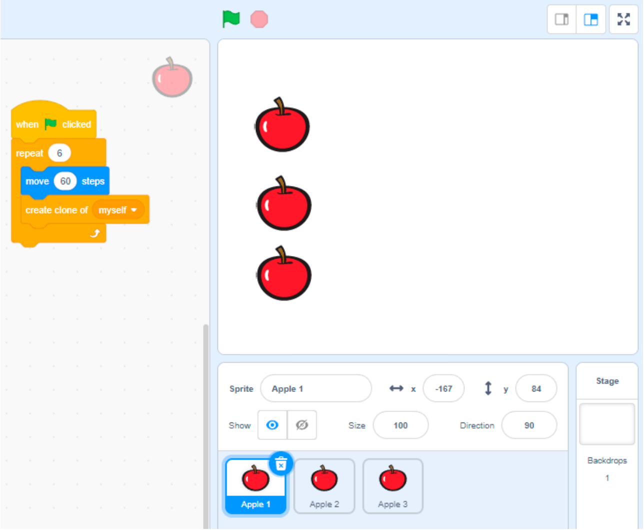
The behaviour of the *sprite* with respect to this code will be to move 60 steps (in this context, one step is one pixel) and make a copy of itself. This behaviour will be repeated 4 times, as defined by the loop. So the code is a representation of the 4 × 1 operation, and the result is shown on the screen after clicking the green flag.

Before clicking on the green flag:	After clicking on the green flag:
	

Even more complex multiplications can be represented by creating more *sprites*.

Note: In the example above, the mathematical goal is to represent the 4 × 1 operation with material, namely apples. The distance between the initial location and the final location of the apple could also be represented by 4 × 60. This flexibility in the way of interpreting the code is one of the great strengths of using code to represent mathematical situations, but it also shows the importance of having a clear instructional intent.

By adding *sprites*, the operation that is represented can be changed. In this example, there are three *sprites*, Apple_1, Apple_2 and Apple_3, each with an identical code.

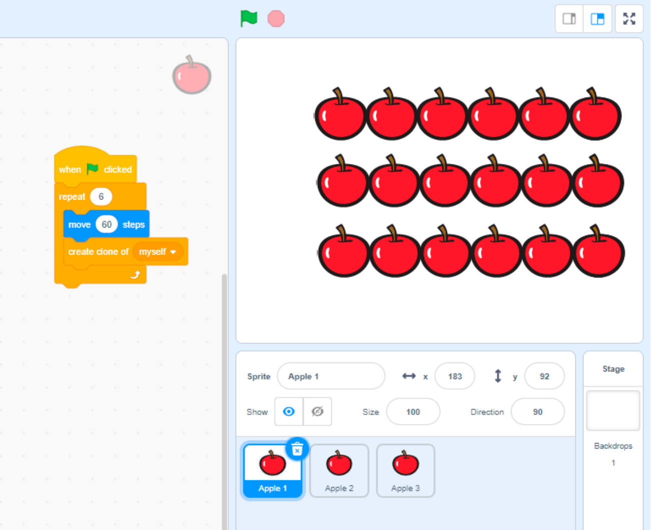


```
when [flag] clicked
repeat 6
    move 60 steps
    create clone of myself
```

Sprite: Apple 1 ↔ x: -167 ↕ y: 84

Show: Size: 100 Direction: 90

When the code is executed by clicking on the green flag, an array of the 3 × 6 operation appears, that is 3 rows of 6 apples.



```
when 🏴 clicked
repeat (6)
    move (60) steps
    create clone of (myself ▾)
```

Sprite: Apple 1, x: 183, y: 92, Size: 100, Direction: 90

In the Python language, the loop is defined with the terms *for* (indicating a number of repetitions) and *while* (indicating a repetition based on a condition). Although the Python language is unlikely to be seen in grade 3, the associated terminology can help to better understand the nature of the loop. In the case of the example above, it is a "for" type loop – the repetition has a defined duration. Conditions will be covered in more detail later in this guide.

This code represents the 3 × 6 operation well with an array, but there is a big disadvantage – it is difficult to alter. To represent the multiplication 4 × 8, for example, one would have to create another *sprite*, Apple_4, and then alter the code so that each *sprite* has 8 repetitions instead of 6. This process can be long, and the structure of the code makes in so debugging can be difficult to accomplish, as the code is in several places. The solution to this dilemma is revealed by having two loops work together.

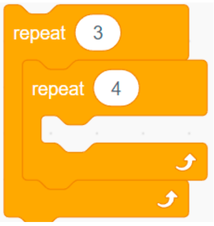
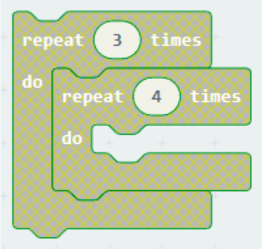
Nested Events

When control structures work together, they are called **nested events**.

Nested events: control structures placed inside other control structures; for example, loops occurring within loops or a conditional statement being evaluated within a loop.

Control structures: line or code block that influences the order in which other code is executed. Control structures affect program flow and include sequential events, repeating events, or selecting whether or not to execute specific lines of code. Sequential events, conditional statements and repeating events (loops) are all control structures.

At first glance, nested events are events that are inside other events. This can look like several different situations depending on the software or coding language chosen.

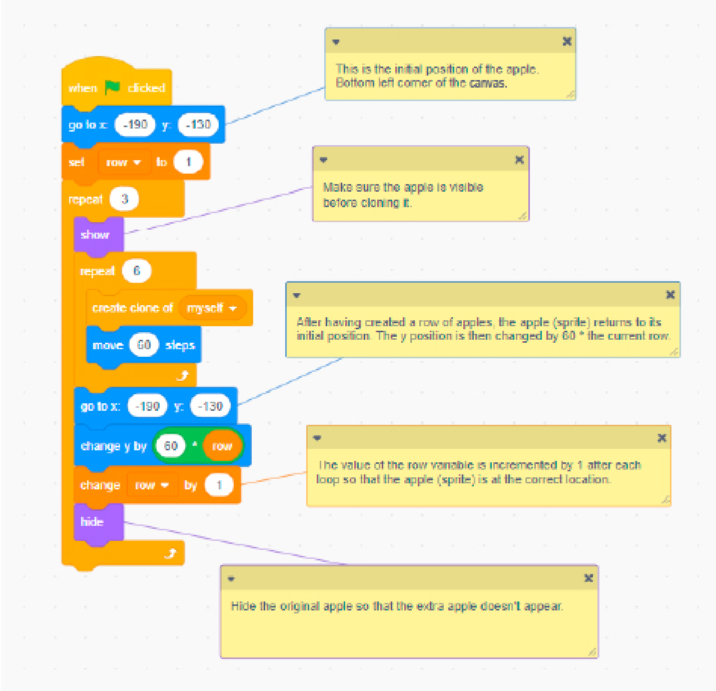
<table>
<tr><th>SCRATCH</th><th>MAKE:CODE</th></tr>
<tr><td>repeat 3
repeat 4</td><td>repeat 3 times
do
repeat 4 times
do</td></tr>
<tr><th colspan="2">PYTHON</th></tr>
<tr><td colspan="2">for index in range(3):
    for index2 in range(4):</td></tr>
<tr><th colspan="2">PSEUDOCODE (EXAMPLE)</th></tr>
<tr><td colspan="2">Repeat (3 times)
    Repeat (4 times)</td></tr>
</table>

Here, the contents of the inner loop will be repeated 4 times. When the operation is complete, the outer loop takes control, which forces the inner loop to start over. This will happen 3 times.

Some similarities can be noticed between block-based coding software and coding languages. The code reads from top to bottom, and indents are used to show code elements that are inside a control structure. This observation could be useful for students to organize their pseudocode well before starting coding.

Going back to the example of multiplying apples disposed in an array, it is then possible to make the number of rows of apples defined by a loop instead of duplicating *sprites*. That way, the code will be contained in one place, which could make debugging easier and reduce where errors could be inserted.

Example of Code (Scratch)



```
when ⚑ clicked
go to x: -190 y: -130
set row ▾ to 1
repeat 3
    show
    repeat 6
        create clone of myself ▾
        move 60 steps
    go to x: -190 y: -130
    change y by (60 * row)
    change row ▾ by 1
    hide
```

This is the initial position of the apple. Bottom left corner of the canvas.

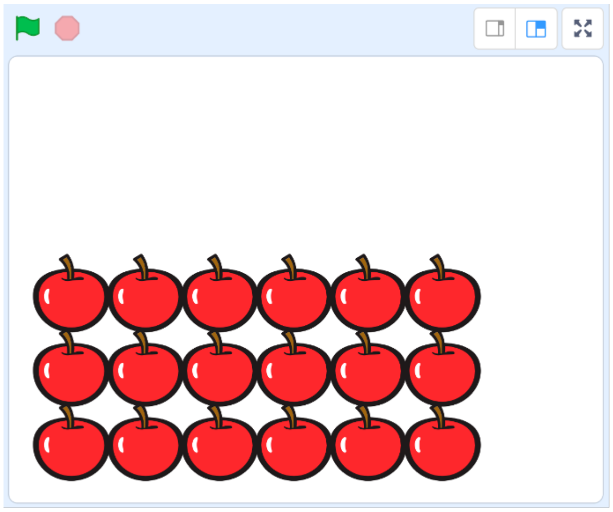
Make sure the apple is visible before cloning it.

After having created a row of apples, the apple (sprite) returns to its initial position. The y position is then changed by 60 * the current row.

The value of the row variable is incremented by 1 after each loop so that the apple (sprite) is at the correct location.

Hide the original apple so that the extra apple doesn't appear.

Outcome



The code now represents a multiplication represented by an array, using a nested loop within another loop. The outer loop represents the number of rows, while the inner loop represents the number of apples per row. The multiplication can therefore be written as follows:

(outer loop repeats) × (inner loop repeats)

Or, in the case of this example,

3 × 6

To go further with this same code, the student could use variables for the two numbers to be multiplied, thus representing any multiplication in an array. An interesting connection can be made here with the area and the perimeter of a rectangle.

Conditional Statements

The transition from sequential events to **conditional statements** is a big step for the student. Conditional statements open the door to all sorts of coding possibilities.

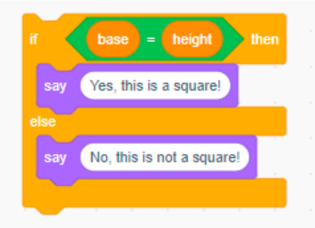
Conditional statements: type of coding instruction that tells the computer to compare values and expressions and make decisions. A conditional statement tells a program to perform an action depending on whether the condition is true or false.

Conditional statements are first present when the execution of an event or a sequence of code depends on the achievement of a specific condition. Boolean values (true or false) are often used and associated with terms such as *if*, *else*, *until* and *when*.

For example, a code representing multiplications using the array could be transformed to determine whether the layout represents a square or not. It would then be the following pseudocode:

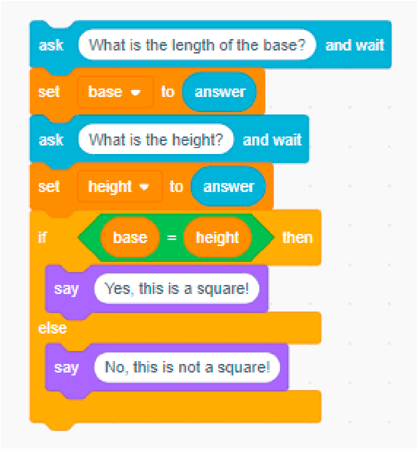
```
If "value_1" = "value_2"
    So
            Say "Yes, it's a square!"
    Else
            Say "No, that's not a square!"
```

Since we are talking about a geometric shape, the names given to "value_1" and "value_2" could be more precise, such as base and height. Here is a sample code representative of this pseudocode:



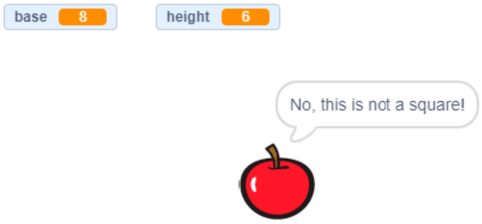
```
if base = height then
    say Yes, this is a square!
else
    say No, this is not a square!
```

In order to make this code work, the variables "base" and "height" must be defined. A few lines of code can be added to ask the user to enter values to determine if it is a square or not.



```
ask What is the length of the base? and wait
set base to answer
ask What is the height? and wait
set height to answer
if base = height then
    say Yes, this is a square!
else
    say No, this is not a square!
```

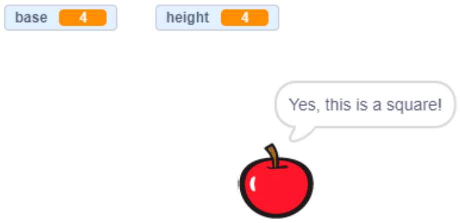
The result: the *sprite* indicates whether the dimensions entered correspond to the definition of a square.



base 8

height 6

No, this is not a square!



base 4

height 4

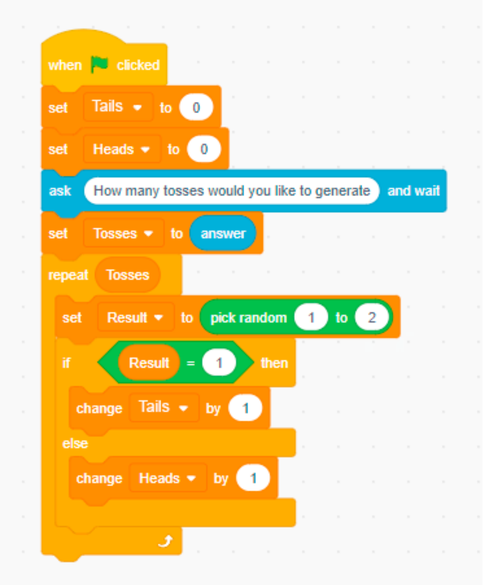
Yes, this is a square!

This is just one example of an infinite number of possibilities. Conditional statements can also use the power of nested events. Instead of the array, it is possible to represent a situation in which the theoretical probability of getting "heads" or "tails" by flipping a coin is determined.

In this context, there are two possibilities, either "heads" or "tails". These two possibilities can be associated with two numbers, 1 and 2, and generate random results that are then compiled in a table. A pseudocode to accomplish this outcome might look like this:

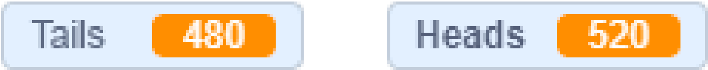
```
Variables
    Tails (set to 0)
    Heads (set to 0)
    Throws
    Results
Ask "How many throws do you want to generate?"
    Throws = Response
Repeat ("Throws") times
    Set "Results" to (random number between 1 and 2)
    If "Result" = 2
        Then add 1 to Tails
    Else
        Add 1 to Heads
```

The corresponding code may look like this:



```
when ⚑ clicked
set Tails ▾ to 0
set Heads ▾ to 0
ask How many tosses would you like to generate and wait
set Tosses ▾ to answer
repeat Tosses
    set Result ▾ to pick random 1 to 2
    if Result = 1 then
        change Tails ▾ by 1
    else
        change Heads ▾ by 1
```

As an example, here is the result generated by this code when 1000 throws have been entered:



Tails 480

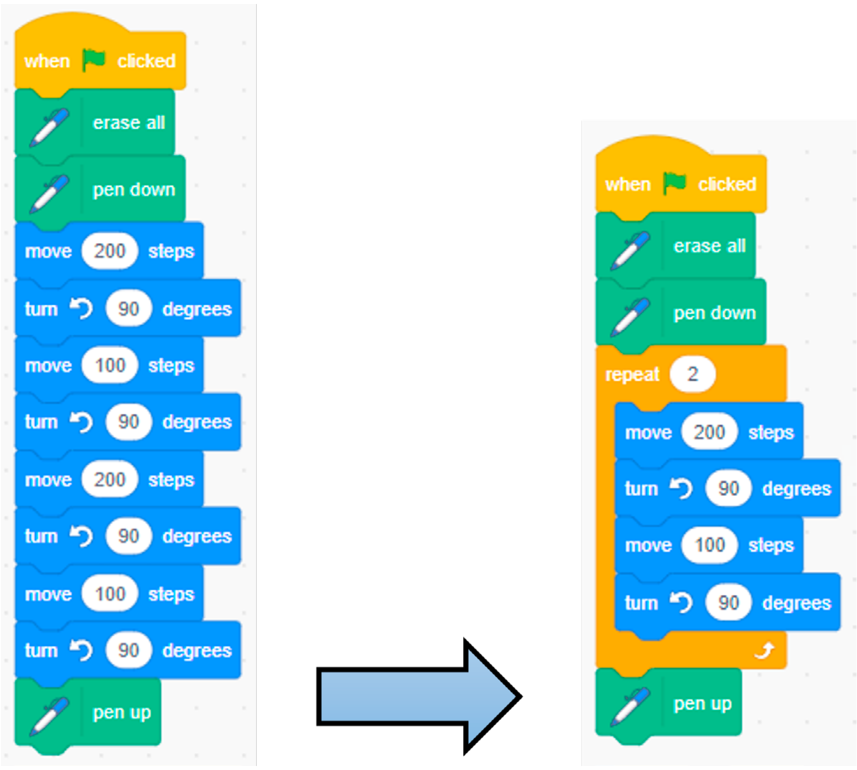
Heads 520

This result was generated in less than a second. Comparing this to the time required to complete 1000 real-time throws, it is possible to see the advantage of using code to solve the problem. In addition, students can now use variables within calculations to compare theoretical probability and experimental probability. Or the student could use this as a basis for determining the experimental probability of another mathematical situation.

Efficiency in the Context of Coding

Adding conditional statements allows the student to represent and solve very complex mathematical problems. It is important, however, to distinguish complexity from code length! A code can be very complex without being very long. It is therefore important to present students with the challenge of producing functional code in a minimum of lines of code. The more a person works on a code, the more elements they add to it, without realizing that there may be a more efficient way to achieve the same outcome. Code that is too long, or too cluttered, can become more difficult to debug and does not lend itself well to teamwork.

There are several ways to make code more efficient. It is possible, for example, to insert a loop when there are repeating code elements. In this example, the desired outcome is to draw a rectangle.



```
when 🏁 clicked
erase all
pen down
move 200 steps
turn ⟲ 90 degrees
move 100 steps
turn ⟲ 90 degrees
move 200 steps
turn ⟲ 90 degrees
move 100 steps
turn ⟲ 90 degrees
pen up
```

```
when 🏁 clicked
erase all
pen down
repeat 2
    move 200 steps
    turn ⟲ 90 degrees
    move 100 steps
    turn ⟲ 90 degrees
pen up
```

Both of these lines of code accomplish the same outcome of drawing a rectangle, but the code on the right does it more efficiently.

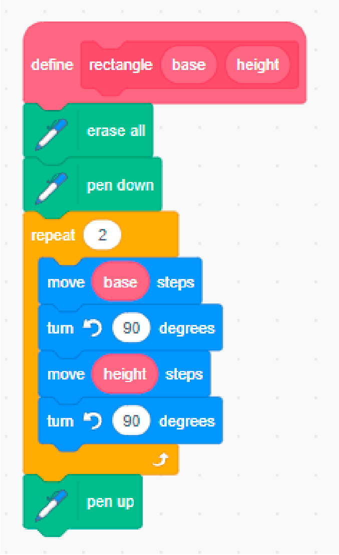
Several text-based and block-based coding software allow you to customize blocks. This ensures that a code sequence can be written only once, but can appear in multiple places in the main code. This is another way to make code more efficient.

Subprograms

A **subprogram** allows you to create a small sequence of code that can be called upon more often. This sequence is not part of the main code. You need a triggering event, a message, to start the subprogram. When the message is received, the entire subprogram is executed before the rest of the main code is executed.

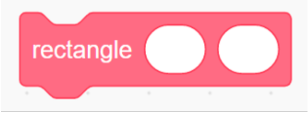
In the example of drawing geometric shapes, it is possible to create a custom block that, when executed, calls a series of specific instructions (a subprogram). This involves creating a block with two adjustable parameters, namely the base and height. In this case, the "rectangle" block sends the message to start executing the code below the "define: rectangle" block. This means that a rectangle of any size can be generated anywhere on the canvas, because the rectangle has been defined. Once the rectangle has been drawn, the main code will continue.

Subprogram: small set of instructions that performs a simple task. Subprograms can be combined into a main program to accomplish an important task using simple steps.



```
define rectangle (base) (height)
erase all
pen down
repeat (2)
    move (base) steps
    turn ⟲ (90) degrees
    move (height) steps
    turn ⟲ (90) degrees
pen up
```

To insert the rectangle, simply add the new "rectangle" block to the code and indicate the values of the base and the height.



```
rectangle ( ) ( )
```

Note: In the Scratch software, it is possible to create personalized blocks from the "My blocks" menu. How to access such subprograms depends on the software or language chosen.

This kind of generalization lends itself well to mathematical modelling, where a model can be defined inside a custom block and then used in code to check if the model is functional. If a model that represents the mathematical situation well is created, the custom block will be a very efficient way to insert the model, which could be very complex, into the main code.

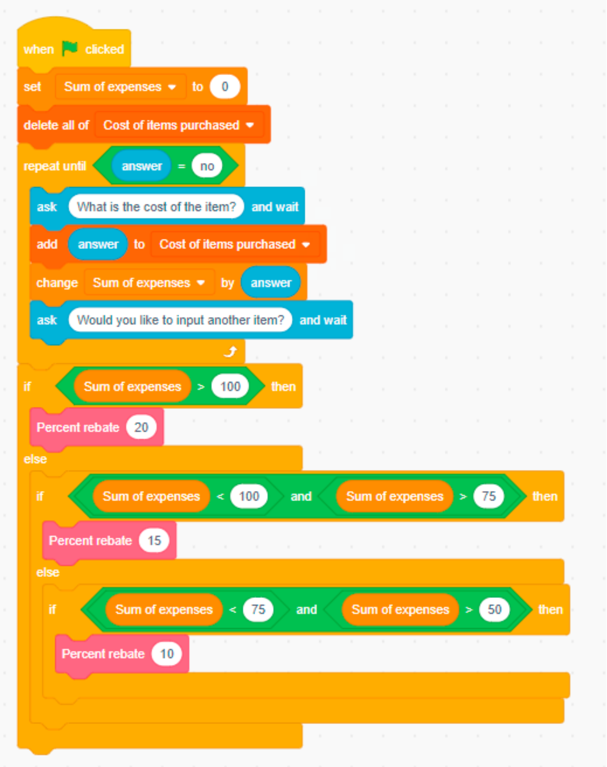
Conditional statements can also play an important role in such code. An example of this integration can be demonstrated in a financial literacy situation, where the intention is to give a discount if a person spends a certain amount. You must first check several parameters so that the code is functional.

- You have to be able to add up the cost of the items to be purchased.
- It is necessary to confirm that all expenses have been entered by ensuring that the condition is met.
- You have to define the conditions to obtain a discount.
- The discount must be calculated and applied to the total.

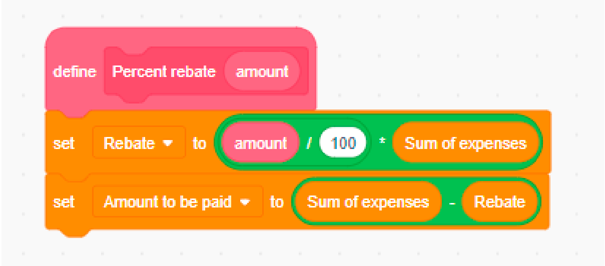
If the exact number of items in the list is known, the number of repetitions in the loop becomes easy to predict. However, if this number is unknown, a **defined count** should be used.

In this case, the "repeat until" block becomes very useful, because it becomes possible to define the situation that ends the loop. An example of code that could therefore meet the needs listed above would be the following:

Defined count:
number of times instructions are repeated according to a predefined value or until a condition is met.



```
when 🏁 clicked
set Sum of expenses ▾ to 0
delete all of Cost of items purchased ▾
repeat until answer = no
    ask What is the cost of the item? and wait
    add answer to Cost of items purchased ▾
    change Sum of expenses ▾ by answer
    ask Would you like to input another item? and wait
                                         ↻
if Sum of expenses > 100 then
    Percent rebate 20
else
    if Sum of expenses < 100 and Sum of expenses > 75 then
        Percent rebate 15
    else
        if Sum of expenses < 75 and Sum of expenses > 50 then
            Percent rebate 10
```



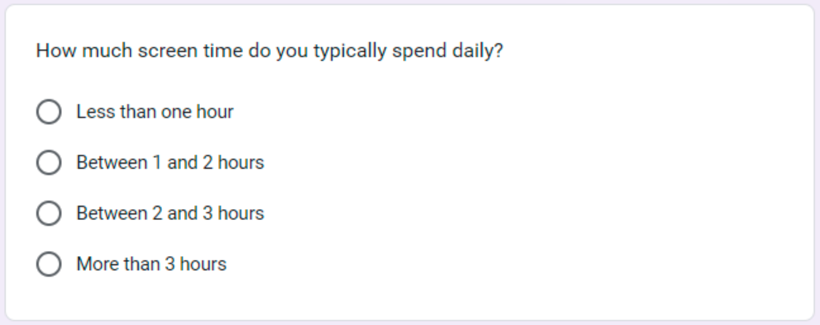
```
define Percent rebate amount
set Rebate ▾ to amount / 100 * Sum of expenses
set Amount to be paid ▾ to Sum of expenses - Rebate
```

This code shows the usefulness of multiple coding structures at the same time. The defined count loop allows the entry of an unlimited number of items, until the conditions are met for its termination. Nested events are used to determine if the conditions for a discount are met. Finally, the custom block "percent rebate" allows us to alter the percentage according to the amount spent.

The student could then add code to calculate the tax or change due. The student could take the total and apply the interest rate of a credit card to determine if it is a good payment method for the expense in question. The student could also add custom blocks for payment terms (monthly, quarterly, weekly, etc.). There are many ways to customize a code like this.

Data Analysis Through Coding

In order to make good decisions, you need good data. Coding software and microcontrollers (Arduino, Micro:bit, etc.) automate the **collection and analysis of data**, which enables decision-making and provides good arguments to support these decisions. Data entry can be done in several ways; for example, code could be written to enable data collection through a form that compiles the data into a spreadsheet. Most spreadsheets accept lines of code to facilitate data analysis.



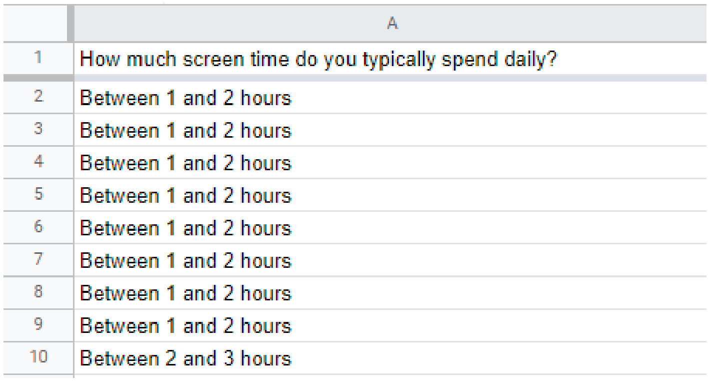
How much screen time do you typically spend daily?

○ Less than one hour

○ Between 1 and 2 hours

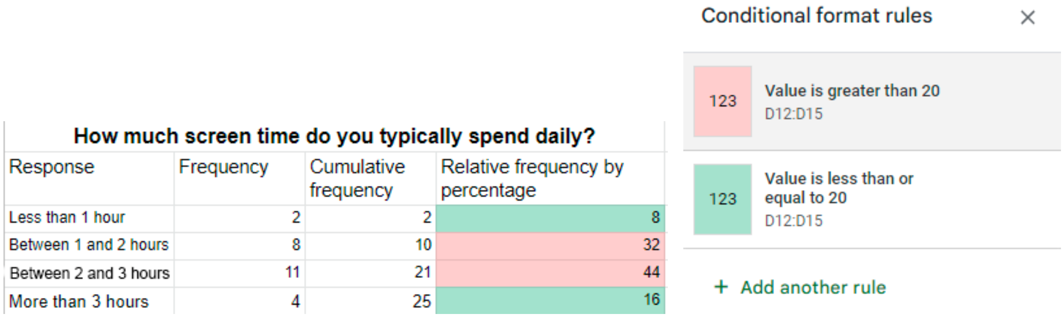
○ Between 2 and 3 hours

○ More than 3 hours



	A
1	How much screen time do you typically spend daily?
2	Between 1 and 2 hours
3	Between 1 and 2 hours
4	Between 1 and 2 hours
5	Between 1 and 2 hours
6	Between 1 and 2 hours
7	Between 1 and 2 hours
8	Between 1 and 2 hours
9	Between 1 and 2 hours
10	Between 2 and 3 hours

Raw data can be turned into a relative-frequency table, and conditional formatting can be applied to cells to highlight specific data or trends.



How much screen time do you typically spend daily?			
Response	Frequency	Cumulative frequency	Relative frequency by percentage
Less than 1 hour	2	2	🟩 8
Between 1 and 2 hours	8	10	🟥 32
Between 2 and 3 hours	11	21	🟥 44
More than 3 hours	4	25	🟩 16

Conditional format rules ✕

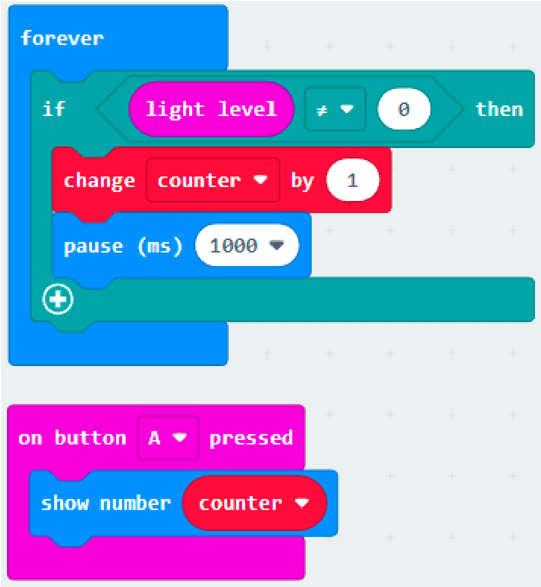
🟥 123 Value is greater than 20
D12:D15

🟩 123 Value is less than or equal to 20
D12:D15

For example, the first rule could be written as follows: "IF (value) > 20, THEN cell color = red."

Note: Regarding the capture of the conditional formatting, you must right-click on the cell and choose "conditional formatting".

Sensors encourage us to go one step further. Many microcontrollers have installed or pluggable sensors that allow data collection; for example, a microcontroller with a light sensor could compile the number of minutes per day the lights are on in a classroom and then determine the electricity cost based on the efficiency written on the bulbs. The code for this task might look like this:



```
forever
    if light level ≠ 0 then
        change counter by 1
        pause (ms) 1000
    ⊕

on button A pressed
    show number counter
```

In this circumstance, one second must be added to the counter for each second that the light sensor detects light energy. A button on the microcontroller is associated with the counter in order to display the total number of seconds on an LED screen. The student should use the trial and error method to determine the appropriate light detection threshold to not have erroneous data.

These microcontrollers are everywhere in a student's daily life – the smart thermostat that acts as a switch that activates the ventilation system when a temperature is reached, or the accelerometer of a smart phone or a smart watch that determines the number of steps a person takes during the day in order to determine the distance covered. Discussions of the ways data is collected and analyzed through coding allow students to further explore the limits of what is possible with code.

Learning Situation – Grade 1

Title: Object Hunting

Duration: 75 minutes

Overview

In this learning situation, the student moves a *sprite* through a forest using block-based programming software. The student also uses their knowledge of addition facts so that each movement of the *sprite* adds up to 10 movements.

This learning situation includes plugged coding activities, that is, coding requiring a coding software, platform or device. It is possible, however, to adapt the activities so that they are unplugged.

Overall and Specific Expectations

Number

B2. Use knowledge of numbers and operations to solve mathematical problems encountered in everyday life.

B2.2 Recall and demonstrate addition facts for numbers up to 10, and related subtraction facts.

Algebra

C3. Solve problems and create computational representations of mathematical situations using coding concepts and skills.

C3.1 Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves sequential events.

C3.2 Read and alter existing code, including code that involves sequential events, and describe how changes to the code affect the outcomes.

Preferred High-Impact Instructional Practices in Mathematics

Learning Goals, Success Criteria and Descriptive Feedback

In this learning situation, the learning goals and the success criteria can be displayed on a large sheet, accessible to the students. Students need to know the expected goals and how to access them. Throughout the lesson, the educator circulates, observes, questions and comments on the work of the students by giving them descriptive feedback based on the learning goals and the success criteria.

Tools and Representations

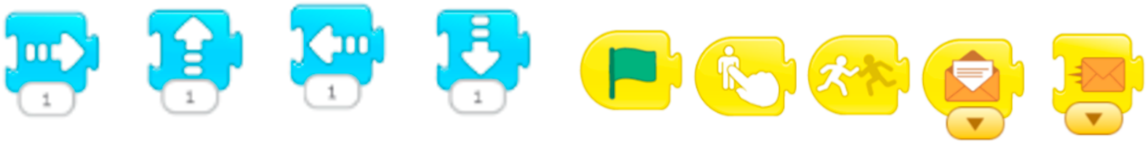
The use of representations of math facts is important in this learning situation. The representation of 10 using a number line, a Rekenrek, a ten frame, a number chart and objects, such as counters, is used so that the student can represent different movements that add up to 10. In addition, the student also knows the related subtraction facts.

Math Conversations

As soon as the Warm-Up begins, the conversation begins. When the educator presents the illustrations, the students question each other and discuss their hypotheses and ideas. Throughout the process, the students explain their translations using blocks.

Prior Knowledge and Skills

- Use equipment, such as a Rekenrek, a ten frame or a beaded line.
- Know the translation blocks and the starting blocks (possibly seen in unplugged coding).



Learning Goals

At the end of this learning situation, the student will be able to:

- perform translations using addition and subtraction facts up to 10;
- create sequential event codes;
- alter codes to make them effective sequential codes.

Criteria According to the Achievement Chart

Knowledge and Understanding

- The student determines the sequential events of a code.
- The student uses addition and subtraction facts up to 10.

Thinking

- The student alters their code to make it more efficient or more original.

Communication

- The student uses the appropriate vocabulary for coding (block, *sprite*).
- The student clearly explains the blocks that are in their code.

Application

- The student creates efficient code that makes their *sprite* move.
- The student uses the variables in their blocks to make the *sprite* move.

Materials

- ten frame
- Rekenrek
- number line
- cubes or any other counters
- beaded line
- two-coloured counters
- block-based programming software
- appendix (Warm-Up)

Note: The examples in this learning situation were created with Scratch Jr.

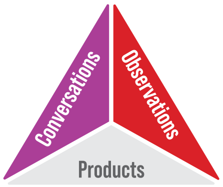
Mathematical Vocabulary

math facts, *sprite*, code, block, sequential events

CONTENTS

Before Learning (Warm-Up) (15 minutes)

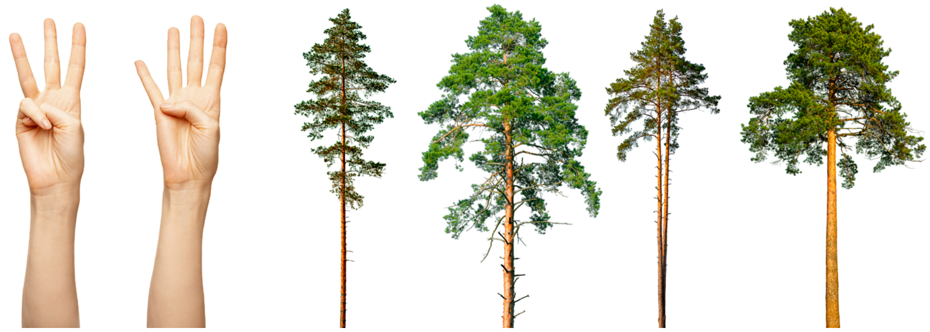
Assessment can be carried out through...



Show students the illustrations below.

Questions to ask

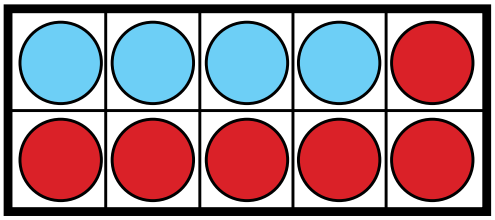
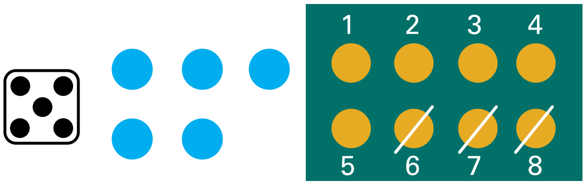
- What do you see?
- How many fingers do you need to add to have 10?
- How many trees do you see?
- How many trees do you need to add to get 10?



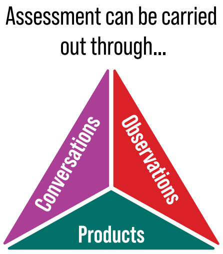
Introduce students to manipulatives, such as the ten frame, two-coloured counters, a Rekenrek, and a beaded line.

Ask students to find different ways to represent 10.

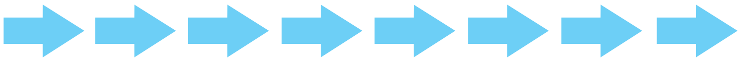
Circulate and observe what students are doing.

POSSIBLE OBSERVATION	POSSIBLE INTERVENTIONS
The student has difficulty representing 10.	Suggest that the student use the two-coloured counters and place them in the ten frame. • How many blue tokens did you place? • How many red tokens did you place?  Remind the student that 10 can be represented with objects, operations, symbols and pictures. Give the student an example for the number 5. • What do you see? • How do you know what that represents? 

Active Learning (Exploration) (45 minutes)

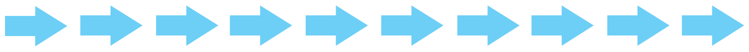


Show students the arrows below and ask them to stand up and make the movements.



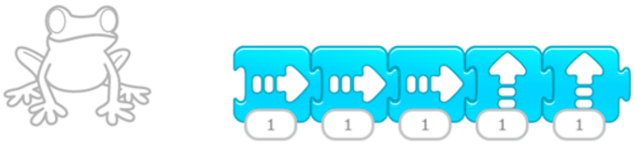
Questions to ask

- How many steps did you take?
- How many arrows do you need to add to make 10 steps altogether?
- Observe the arrows below. How many moves to the left do you have to make to get a total of 8?



Ask the students to create a sequential set of instructions in block-based programming software. *Sprite* movements should represent addition facts up to 10 or associated subtraction facts.

Show students the following code.



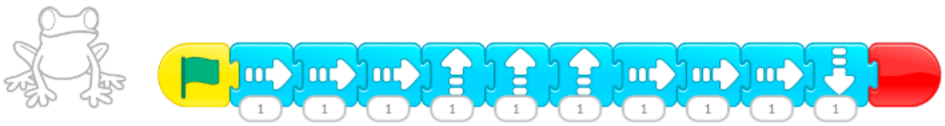
Questions to ask

- What will the frog do?
- How many moves do you need to add for it to make 10 moves in all?
- What is the code missing for it to be functional?

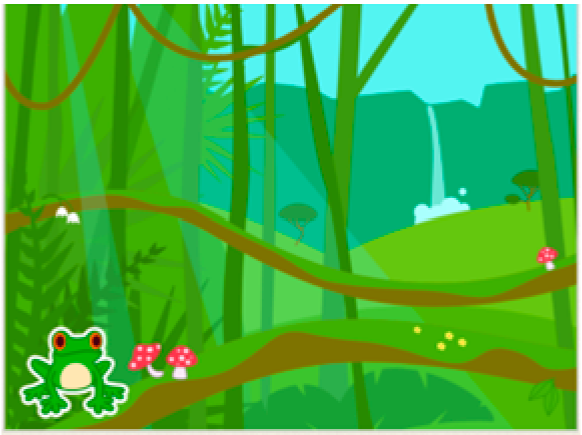
Guide the discussion and get the students to come up with a functional code that includes 10 moves.

Possible answer

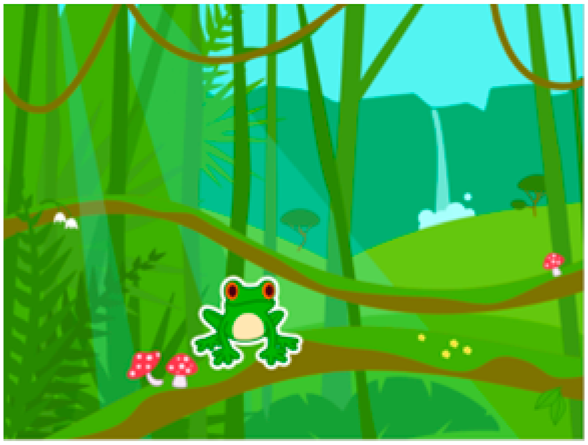
Now here is the code for the frog to make 10 moves. Student suggests adding 1 up arrow, 3 right arrows and 1 down arrow. The addition of the start block and the stop block is also necessary for the code to work.



Here is the position of the frog before the moves:

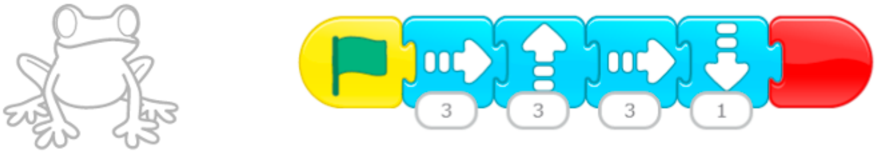


Here is the position of the frog after the moves:



- Bring out the sequential events of the code.
- Ask students if there is a way to alter the code to make it more efficient. (In this context, "more efficient" could be interpreted as using fewer blocks to achieve the same outcome.)

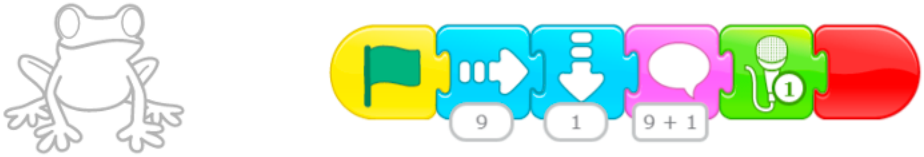
Here is an example of a more efficient code. Variables must be added below the blocks. The code remains a series of sequential events.



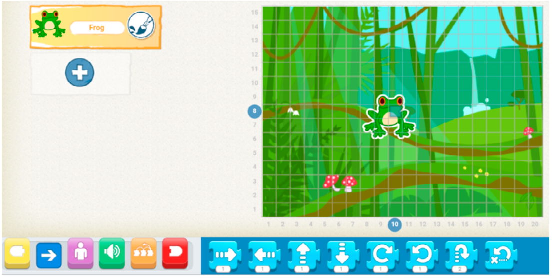
- Suggest that the students create other sequential sets of instruction for their frog in the forest using block-based programming software.
- Tell students that the moves of the frog must represent addition or subtraction facts for the number 10.

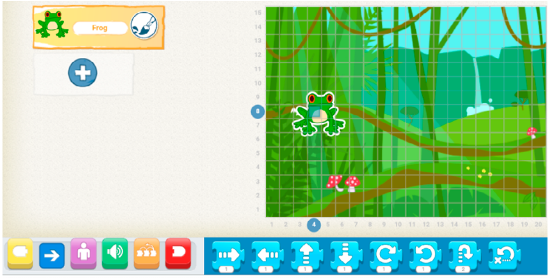
Possible answers

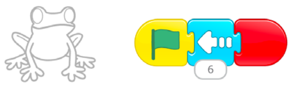
The student can then add a green block in which the answer is dictated; for example, the student says, "Nine plus one equals ten".

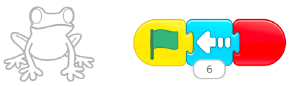


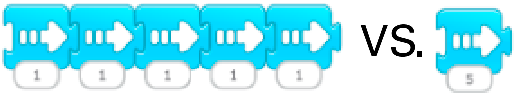
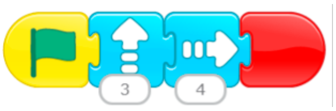
The frog is placed on the grid at 10. The student creates the code below so that the frog is at 4.



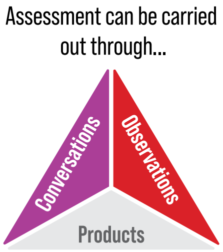






POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student is unable to get their code to work.	Which block should you use to start your code?
The student uses several blocks rather than changing the variable of their block.	What can you do to make your code more efficient? Do you notice repetitions in the blocks of your code? 
The student's instructions do not represent 10.	How many upward movements did the frog make? Represent these using the Rekenrek beads. How many movements did the frog make to the right? Represent these using the Rekenrek beads.  Does the frog's total moves represent 10 beads, or a full Rekenrek row? If not, how many beads do you need to add?

Review (15 minutes)



Ask students to present their code to the class. Invite other students to try them.

Observe the code with the students and highlight the sequential events in the code.

Review the blocks and their meanings with the students.

Consolidation of Learning

Suggest that the students do a treasure hunt for objects using robots such as Bee-Bot, Sphero, Ozobot or others. If no robotic device is available, online interactive versions exist for several educational robots.

CONSIDERATIONS

Links to Other Curriculum Expectations (Sequential Events)

Number

B2.2 Recall and demonstrate addition facts for numbers up to 10, and related subtraction facts.

Using a variety of move combinations that add up to 10, the student can make a *sprite* move to specific locations. Subtraction can be represented by movements in the opposite direction (for example, 10 steps to the left and 4 steps to the right is a movement of 6 steps, which could represent 10 – 4 = 6).

Algebra

C1.2 Create and translate patterns using movements, sounds, objects, shapes, letters, and numbers.

The student can create a repeating pattern using the blocks in the coding software, then share their code with a classmate to determine the rule and extend the pattern.

Data

D1.1 Sort sets of data about people or things according to one attribute, and describe rules used for sorting.

The student can have a collection of *sprites* on their canvas and classify them according to their attributes using a sequential set of instructions so that each *sprite* has its designated place.

Spatial Sense

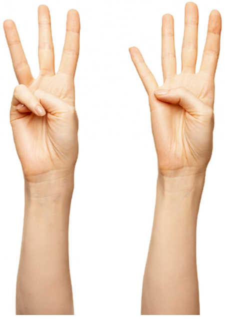
E1.5 Give and follow directions for moving from one location to another.

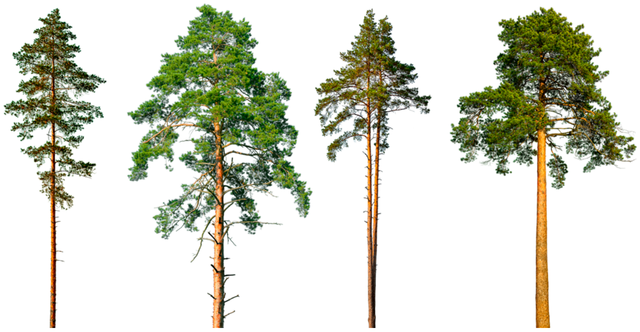
Using the vocabulary associated with translation blocks (left, right, up, down), the student creates a path for a *sprite* comprising of a sequence of movements.

Differentiated Instruction and Universal Design for Learning

- Give a copy of the blocks to the students and ask them to solve the problem in unplugged coding.
- Provide an example that has already been done for the student to build on.
- Ask the students to use their 10 movements to get the *sprite* to reach various objects in the forest.
- Suggest that the students add pages to their project in order to move their *sprite* from one page to another.

Appendix – Warm-Up





Learning Situation – Grade 2

Title: An Obstacle Course

Duration: 75 minutes

Overview

In this learning situation, the student reads, alters and creates codes in order to navigate a *sprite* by means of an obstacle course. The student also describes the relative position of an object and the movements required to move from one object to another when creating the obstacle course.

This learning situation includes plugged coding activities, that is, coding requiring a coding software, platform or device. It is possible, however, to adapt the activities so that they are unplugged.

Overall and Spectific Expectations

Algebra

C3. Solve problems and create computational representations of mathematical situations using coding concepts and skills.

C3.1 Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves sequential and concurrent events.

C3.2 Read and alter existing code, including code that involves sequential and concurrent events, and describe how changes to the code affect the outcomes.

Spatial Sense

E1. Describe and represent shape, location, and movement by applying geometric properties and spatial relationships in order to navigate the world around them.

E1.5 Describe the relative positions of several objects and the movements needed to get from one object to another.

Preferred High-Impact Instructional Practices in Mathematics

Math Conversations

During Warm-Up, a math conversation is initiated with the students in the class. This conversation encourages them to share their ideas with others, to show their understanding of the relative position of an object and translations, to reason and to prove their reasoning. Students are able to make connections to real life experiences and to communicate their knowledge related to the topic at hand. Evidence of student achivement can be collected to inform instruction.

Flexible Groupings

Working in teams of two, when creating and altering code, especially encourages communication and collaboration. Flexible groupings promote collaboration and encourages students to take part in rich math conversations. Students can learn from each other. In this way, students develop their mathematical thinking. While taking into account the profile of the students, the groupings can be random, heterogeneous or homogeneous, depending on the needs.

Small-Group Instruction

Small-group instruction is a practice to advance student learning. Throughout the creation and alteration of a code, it is important to circulate among the groups of students in order to provide them with the necessary spontaneous support according to what is observed and heard. Small-group instruction provides an opportunity to review mathematical concepts with students that support their new learning. These students are therefore able to deepen their understanding of the concepts under study (position and translation or coding).

Direct Instruction

When a code does not give the expected outcome, educators could ask students to explain it. This is a good time to offer direct instruction related to the difficulty observed. This instruction verifies student understanding and provides immediate feedback.

Prior Knowledge and Skills

It would be better if the students have the opportunity to explore the coding software to be used in order to know the interface and its basic elements. However, this learning situation can also serve as an initial exploration of a software by allocating more time, so that the students become familiar with it in the process.

Learning Goals

At the end of this learning situation, the student will be able to:

- use terminology specific to an object's relative position *(above, below, on, under, beside, inside, outside, right, left)* and coding *(sprite, sequential events, concurrent events)*;
- describe the movements of a *sprite (to the right, to the left, up, down);*
- writing code that involves moving more than one sprite at the same time;
- read code, then alter it to make it more efficient.

Criteria According to the Achievement Chart

Knowledge and Understanding

- The student knows the appearance and function of the blocks in the coding software used.
- The student describes movements from one object to another by referring to the direction and the number of movements.
- The student determines the blocks and actions of each block used in the sequential and concurrent events.
- The student understands what two sets of concurrent instructions represent.

Thinking

- The student plans the location and duration of a "pause" block in order to properly synchronize the concurrent events in their code.
- The student predicts the result of a code and alters it to achieve a specific outcome.
- The student alters a code and determines the effect of the changes on the behaviour of a *sprite*.

Communication

- The student explains their code using the appropriate terminology *(left, right, top, bottom,* etc.).
- The student uses vocabulary related to the relative position of an object *(above, below, on, under, beside, inside, outside, to the right, to the left)*.

Application

- The student puts the blocks in the right places and in the right order to create code that contain concurrent events.
- The student demonstrates an understanding of the difference between sequential and concurrent events in the context of coding.

Materials

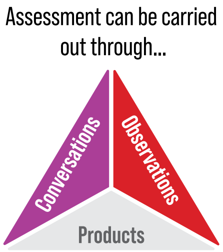
- block-based programming software appropriate to the skill of the student
- graph paper and pencils (optional)
- appendix 1 (Warm-up)
- appendix 2 (Butterfly and Bird Codes)

Note: The examples in this learning situation were created with Scratch Jr.

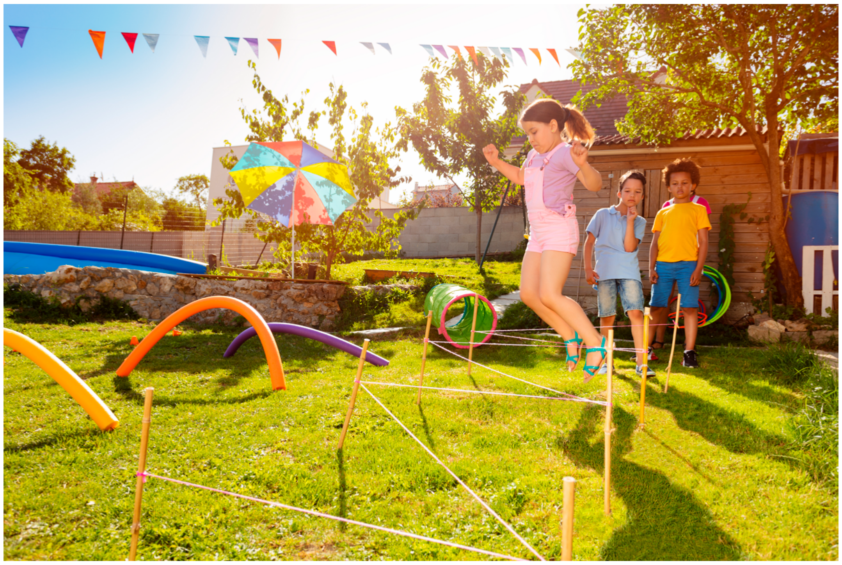
Mathematical Vocabulary

code, *sprite*, sequential events, concurrent events, left, right, above, below, front, behind, translations (units left, right, up, down)

Before Learning (Warm-Up) (10 minutes)



Show students the picture below appendix 1 (Warm-up) and ask them, "What do you notice?" Use the Think-Pair-Share strategy to initiate a math conversation with students.



Think

Require at least one minute of silent reflection. This ensures that students think without the pressure of time or the influence of others' answers.

Pair

Ask students to discuss, in pairs, their first impressions of the picture.

Share

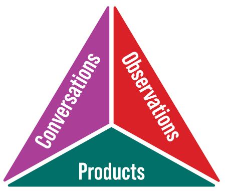
Open the conversation. First, invite the students to express their impressions of the picture as well as their observations following their conversation with their partner. Then, try to direct the questioning towards the mathematical concepts to be studied.

Questions to ask and possible responses from students:

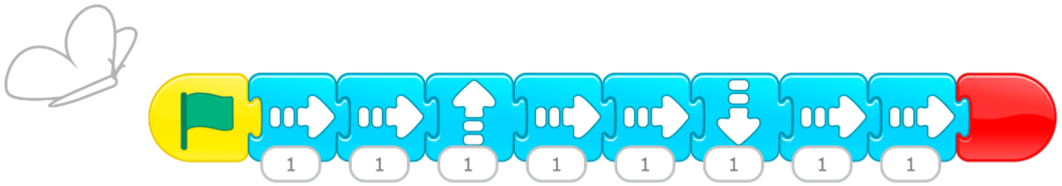
- What are the children doing? What is this type of activity called?
  - An obstacle course.
- Can you describe the obstacle course to me? What do you see?
  - A child jumps over a rope.
  - There is a tunnel behind the rope. It is to the left of the two children standing back. There are flags tied to the trees, above the children.
- How many children are racing?
  - Three children are racing.
  - One child is racing and the other two are waiting their turn.
- How can we determine who will win?
  - Time the children to determine who completed the race as quickly as possible.
  - Ask the children to race at the same time. The student who arrives first, wins the race.

Active Learning (Exploration)
(50 minutes in addition to the time allocated to explore the coding software, if necessary)

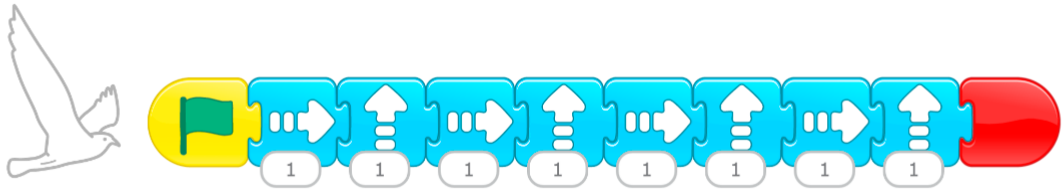
Assessment can be carried out through...



Introduce students to the butterfly code below, either by displaying it on a screen or using color photocopies.



Form teams of two and have students study the code to describe the movement of the butterfly. When the students are satisfied with their answer, ask them to do the same exercise using the bird code below. This time, since the teams may move at different paces, it would be best to distribute color photocopies made on strips of paper.



Afterwards, facilitate a class conversation to compare the two codes.

Questions to ask and possible responses from students:

- Who travels the farthest, the butterfly or the bird?
    - The butterfly travels the farthest, as it moves 6 squares to the right, while the bird only moves 4 squares to the right.
    - The bird travels the farthest, as it moves 4 squares up, while the butterfly moves one square up and then one square down.
- Who will have completed their moves first?
    - Both animals have 8 blocks in their code, so both will complete at the same time.
    - The bird only has 4 move blocks to the right, so it will end its moves before the butterfly.
- What does the green flag mean in the code?
    - This is the starting signal.

Allow students access to their chosen block-based coding software to recreate the code and check their answers to the questions.

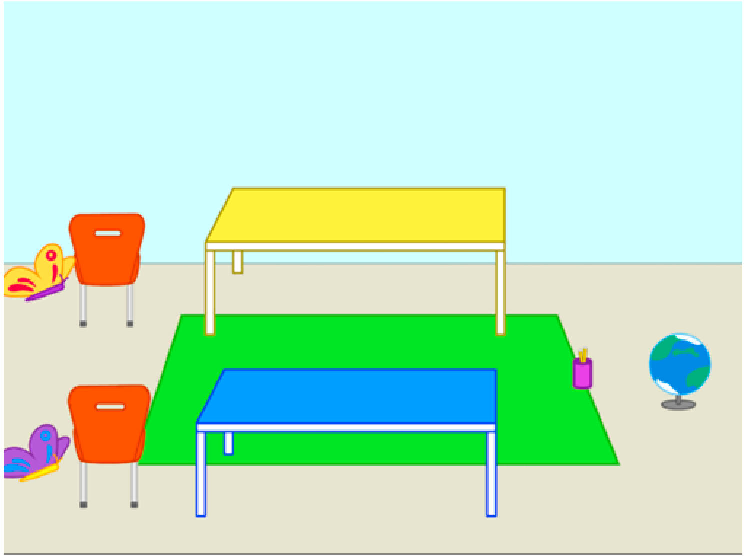
Afterwards, allow the students some time to experiment with the software in order to alter the given code to change the behaviour of the butterfly and that of the bird. This gives students the opportunity to experience the effect of changing, adding and omitting certain code blocks.

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student code does not look like the first code shown at all.	Ask the student to explain aloud the code blocks and their effect. Provide the student with a hard copy of the code to be duplicated so that they can compare it with their code.
The student's code includes logic errors (the code does not give the desired outcome).	Provide the student with a sheet of grid paper so that they can trace the movements and determine the blocks causing the error (it looks like pseudocode). Suggest that the student share their code with another person who could proofread it and give them possible solutions (such as revision in the writing process).
Student code includes syntax errors (code does not work at all).	Review with the student the initial situations (for example, the green flag). Ask the student to check if the code blocks are well connected.

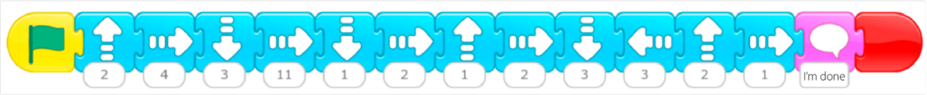
Now, give students a more specific intent for their code. An obstacle course must be created, in which two *sprites* compete, much like the butterfly and the bird. The students choose the two *sprites* as well as the obstacles of the race and describe the movements of the *sprites* related to the obstacles. One of the *sprites* will have to reach the finish line before the other.

Possible answer

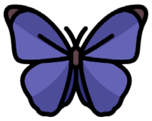
Route: The butterflies will have to fly over the chair, under the table, touch the purple pencil holder and go around the globe to finish in front of it. The winning butterfly will be the first to shout "I'm done!".

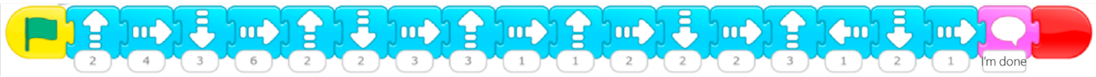


Possible code for the yellow butterfly:



Possible code for the purple butterfly:

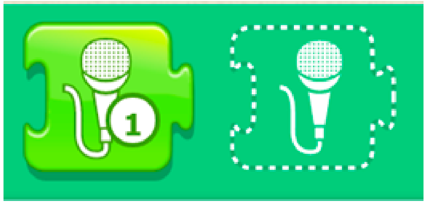




Note: It is important to encourage students to execute their code often. A lot of learning can happen by trial and error.

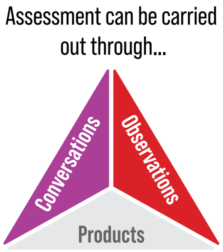
POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student does not know where to start.	Have the student choose the obstacles for the race from the *sprites* menu, then determine the route. Provide the student with concrete materials with which a representation of the obstacle course could be constructed, in order to have a visual cue before beginning to code. Ask the student to write, on paper, a pseudocode, such as a description or a drawing of their obstacle course. Grid paper could be used.
The student does not deviate at all from the code provided at the beginning of the activity.	Ask the student to describe the obstacle course. Do the movements of the *sprites* reflect the race? How can the movements be made more realistic? Encourage the student to alter the blocks, one at a time, and execute the code with each alteration in order to see the effect on the behaviour of the *sprites*.
The student creates two identical codes for their two *sprites*.	**Note:** It is very common for the student to create two identical codes for their two *sprites* when talking about concurrent events, believing that the two *sprites* must do the same thing at the same time. Ask the student if it is possible to determine which is the winning *sprite* when the two codes are identical. Use the example of a different kind of race to activate the student's prior knowledge. How do you know when a race has started? A green light, a whistle, a sound? Once the race has started, do the *sprites* move exactly the same way?

Note: Several coding software programs allow students to record their voice, which gives them the opportunity to create a narrative for the race and, also, to communicate their understanding of the concepts orally rather than in writing.



Example of a block allowing voice recording in Scratch Jr.

Review (15 minutes)



- Choose a few codes to present to the class. The choice can be made by asking for volunteers or by strategically selecting codes that emphasize the instructional intent of familiarizing students with the relative movements and positions of objects and concurrent events.
- Ask the chosen teams to present their code to the class. The team must explain its code as well as the rules of its obstacle course (the movements to be made and how to win). Then, the students in the class analyze the code to try to predict who will win the race. Finally, the code is executed to find out which *sprite* will win.
- Ask students the following questions:
  - How do we ensure that both codes start at the same time? (By clicking on the green flag.)
  - What would happen if we clicked on each of the *sprites* to execute the code? (You would have to move the mouse pointer from *sprite* to *sprite* to activate them, so it would be impossible to activate them at exactly the same time.)

Example of a block allowing to start a sequence by clicking the *sprite*.

- In the case where we have to click on each of the sprites to execute the code, do the events remain concurrent? (Even though the two codes are not executed at the same time, there are times when both *sprites* move at the same time. These are concurrent events at those times.)

Consolidation of Learning

Robotics is a great way to consolidate coding learning. A race in which a robot, such as Sphero, Ozobot, LEGO or other, competes against another robot with a different code could make the concept of concurrent events even clearer for students.

Unplugged Option

If access to the material is limited to computer-based platforms, it would be possible to emphasize concurrent events by asking students to move *sprites* in diagonal paths (for example, to draw a triangle). Diagonal movement requires a horizontal movement and a vertical movement concurrently, because there is no specific block for this kind of movement. The diagonal move is more time efficient than a horizontal move followed by a vertical move, even if the end position is the same. This could make eventual "obstacle races" more interesting.

CONSIDERATIONS

Links to Other Curriculum Expectations (Concurrent Events)

Number

B1.5 Describe what makes a number even or odd.

By using an unknown quantity of *sprites*, the student can write code allowing the *sprites* to be placed, two by two (concurrently), in an organized manner, on the canvas. The student can determine whether the number of *sprites* is even or odd by observing whether or not there are any left after pairing.

B2.6 Represent division of up to 12 items as the equal sharing of a quantity, and solve related problems, using various tools and drawings.

Using a number of *sprites* from 1 to 12, the student can write code in which the objects are put into equal groups; for example, if there are 6 *sprites* on the canvas and the goal is to create 3 equal groups, the student creates code for 3 *sprites* to move concurrently to three different regions of the canvas, then does the same for the other three *sprites* to create 3 groups of 2 *sprites*.

Spatial Sense

E1.5 Describe the relative positions of several objects and the movements needed to get from one object to another.

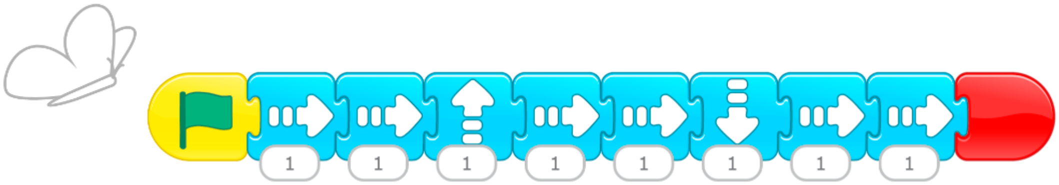
The student can write code so that a *sprite* moves on a map. Diagonal movements require concurrent events, namely a horizontal movement at the same time as a vertical movement.

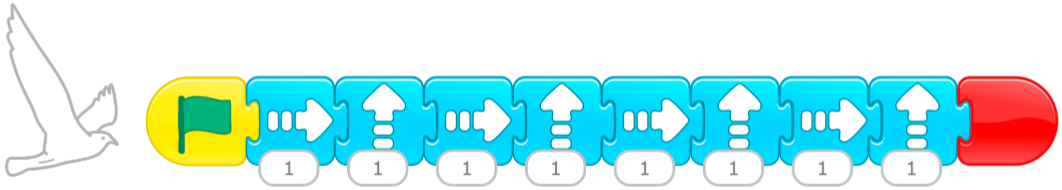
Differentiated Instruction and Universal Design for Learning

- A checklist of the blocks and their function or the colour code used by the coding software could be useful for students who do not yet have a good knowledge of the software.
- Coding an obstacle course could be done in teams of two or three (avoid teams of more than three in order to encourage the participation of each student).
- Have the student attempt to create a race with three *sprites* or a race where the code for each *sprite* is different, but the *sprites* finish the race at the same time.

Appendix 1 – Warm-up

Appendix 2 – Butterfly and Bird Codes





Learning Situation – Grade 3

Title: The Endless Cookies
Duration: 140 to 150 minutes

Overview

Students will be able to practice the multiplication facts as well as the array model by creating code for the baker's machine.

Overall and Spectific Expectations

Number

B2. Use knowledge of numbers and operations to solve mathematical problems encountered in everyday life.

B2.4 Demonstrate an understanding of algorithms for adding and subtracting whole numbers by making connections to and describing the way other tools and strategies are used to add and subtract.

B2.6 Represent multiplication of numbers up to 10 × 10 and division up to 100 ÷ 10, using a variety of tools and drawings, including arrays.

Algebra

C3. Solve problems and create computational representations of mathematical situations using coding concepts and skills.

C3.1 Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves sequential, concurrent, and repeating events.

C3.2 Read and alter existing code, including code that involves sequential, concurrent, and repeating events, and describe how changes to the code affect the outcomes.

Preferred High-Impact Instructional Practices in Mathematics

Learning Goals, Success Criteria and Descriptive Feedback

Post the success criteria in the classroom for students to refer to as they work. Throughout the activity, take the time to circulate in the classroom and observe what the students are doing. Interview them about their work and give them descriptive feedback related to the posted success criteria.

Math Conversations

During Warm-Up, students have the opportunity to discuss possible solutions to the code. During this discussion, students are able to listen to the opinions of other students, explain their reasoning and add ideas to those of others.

Tools and Representations

Students practice their knowledge of multiplication and solidify their learning by representing it using arrays. Various situations can be modeled visually and compared with other representations. Effective use of tools and representations means that different representations are valued and connections will be made between them.

Prior Knowledge and Skills

- Recall multiplication facts up to 10.
- Be able to represent multiplication using an array.
- Be able to represent equal groups with a remainder.
- Use algorithms for adding and subtracting whole numbers.
- Have a basic understanding of the coding software of choice:
    - using the repeat block for definite and indefinite loops;
    - *sprite* cloning;
    - returning to the starting point (without knowing the coordinates);
    - the use of variables.

Learning Goals

At the end of this learning situation, the student will be able to:

- write and execute code to represent the array model;
- add one or more loops in the code to make it more efficient;
- use multiplication facts to help arrange pastries efficiently on a baking sheet.

Criteria According to the Achievement Chart

Knowledge and Understanding

- The student knows the blocks of block-based programming software and understands how they work.
- The student uses the array to represent multiplication facts up to 10 × 10.

Thinking

- Student uses code to represent multiplication using the array.
- The student uses addition and subtraction algorithms to create arrays of prime numbers.
- The student identifies repeating events and places them in a loop to make the code easier to read and alter (efficiency).
- The student alters an erroneous code so that it works.

Communication

The student uses repeating blocks (loops) to make their programming more efficient.

Application

- The student corrects an erroneous code by using their knowledge of the functions of the blocks in the coding software used.
- The student makes connections between the arrangement of the coding blocks and the number facts (for example, a loop can represent repeated addition or multiplication).

Materials

- computer and application or coding software of your choice
- appendix (Warm-Up)

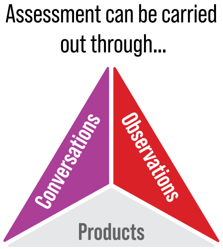
Note: The examples in this learning situation were created with Scratch.

Mathematical Vocabulary

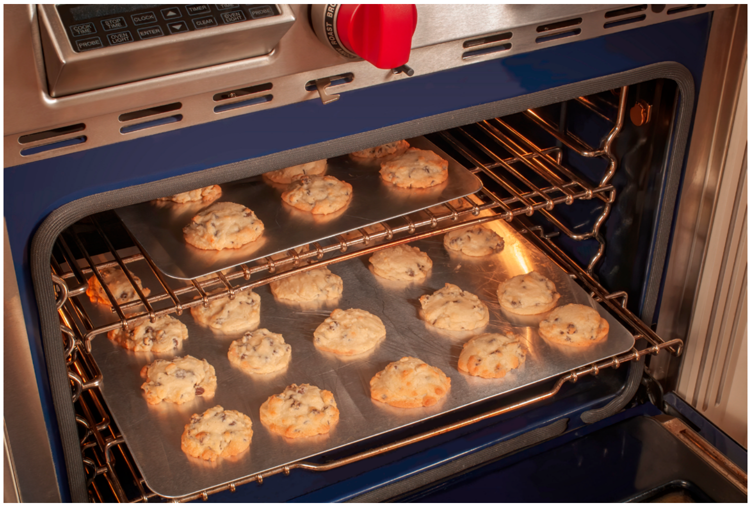
array, code, sequential events, concurrent events, repeating events, loop, variable

CONTENTS

Before Learning (Warm-Up) (30 minutes)



Show the students the picture below . Invite them to share their observations and questions about the photo (I notice, I wonder).



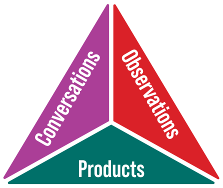
Examples of questions or comments:

- How many cookies are there altogether?
- How can you determine the number of cookies without counting them one by one?
    - I see a baking sheet that has 4 rows of 5 cookies and a baking sheet that has 2 rows of 3 cookies.
    - I believe there are 20 cookies on the bottom baking sheet, even though I can't see the top row, because 4 × 5 = 20.

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student does not know where to start.	What should you do now? What do you see?
The student is unable to make connections with their prior knowledge.	What do you notice in the picture that makes you think of math? Do you know how many cookies there are? How do you know?
The student has difficulty expressing their observations, using mathematical vocabulary.	Could the number of cookies be represented using repeated addition? Could this repeated addition be represented by multiplication? On the bottom baking sheet, how many rows of cookies are there? How many cookies are there in each row?

Active Learning (Exploration) (100 minutes)

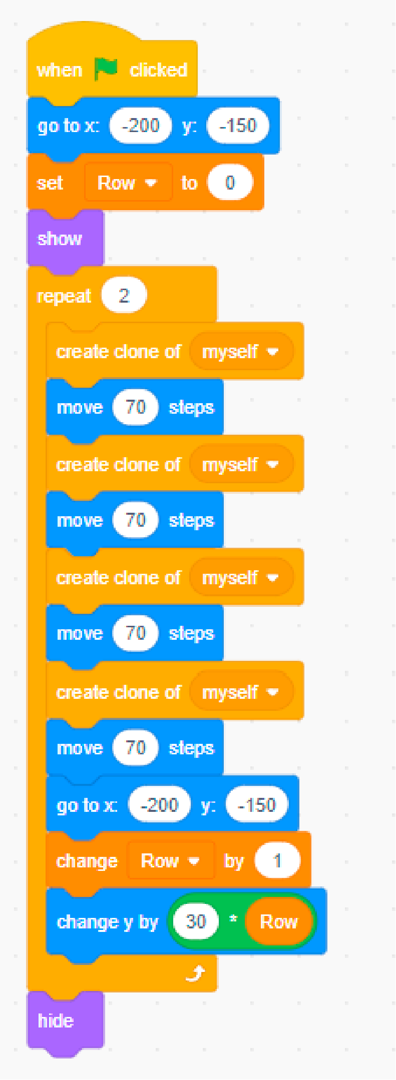
Assessment can be carried out through...



Present the students with the scenario below and the code:

At the bakery, Maxime has to plan where to put the pastries on the baking sheets before baking them. The machine that places pastries on a baking sheet uses block-based coding software. Maxime creates the code below to put 24 pastries on a baking sheet.

Example of Maxime's code:



```
when ⚑ clicked
go to x: -200 y: -150
set Row ▾ to 0
show
repeat 2
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    go to x: -200 y: -150
    change Row ▾ by 1
    change y by (30 * Row)
↻
hide
```

Questions to ask students:

- What do you notice?
- What are we trying to do with this code? (Place 24 pastries on a baking sheet.)
- How many pastries do you think will be on the baking sheet? (There will be 8 pastries on the baking sheet.)
- Which block "puts" a pastry on the baking sheet?
  How many times is it present in the code?

  The "create a clone of myself" block puts the pastry on the baking sheet. This block is present 4 times, but in a loop of 2 repetitions, so 2 x 4 = 8.

- What questions do you ask yourself?
- Could you tell me, in your own words, what this code means?

Invite students to create pseudocode that represents Maxime's code so they can predict what happens when the code is executed. Then, ask them if Maxime's code will produce the desired outcome.

Examples of student thinking:

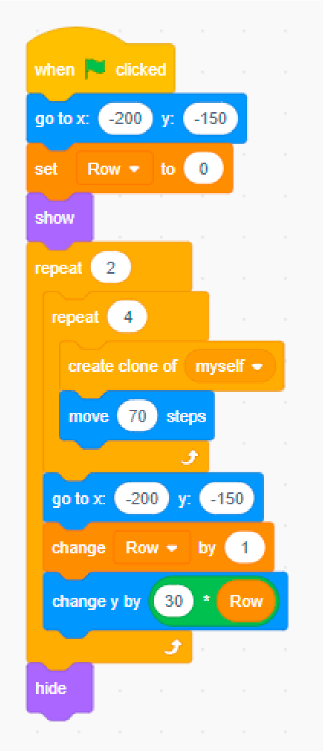
- I notice a loop, meaning a series of blocks are repeated.
- I notice that variable blocks are used. I have never used variables.
- I think 8 objects will be created because the loop repeats twice. In the loop, there is the "make a clone" event which repeats 4 times, 2 x 4 = 8.
- I wonder how I will create a variable.

Example of possible pseudocode:

```
Go to the starting position.
Set the "row" variable to zero.
Show object.
Repeat 2 times.
    Go forward 70 steps.
    Make a copy of the sprite.
    Go forward 70 steps.
    Make a copy of the sprite.
    Go forward 70 steps.
    Make a copy of the sprite.
    Go forward 70 steps.
    Make a copy of the sprite.
    Return to starting position.
    Add 1 to the row variable, so up by 1.
    Add 30 × ("row" value), so 30 × 1*.
Hide the object.
```

* This part of the code moves the *sprite* to the beginning of the correct row. In this case, each row is spaced 30 pixels apart. So the first row (row = 0) starts at the start position, the second row (row = 1) starts at the start position + 30, the third row (row = 2) starts at the start position + 60 , etc.

Note: Students may notice that the part "Go 70 steps forward, make a copy of the *sprite* " repeats 4 times, and ask if it would be possible to use a loop to simplify the code. It is in fact possible. This kind of structure is further developed in Grade 4 (nested events). That being said, here is what the code containing this structure could look like, in case it is one of the avenues of questioning for the students.



```
when 🏴 clicked
go to x: -200 y: -150
set Row ▾ to 0
show
repeat 2
    repeat 4
        create clone of myself ▾
        move 70 steps
    ⤴
    go to x: -200 y: -150
    change Row ▾ by 1
    change y by (30 * Row)
⤴
hide
```

- Have students recreate the code using the chosen software.
- Invite the students to modify Maxime's code so that it includes 24 pastries, as initially desired.

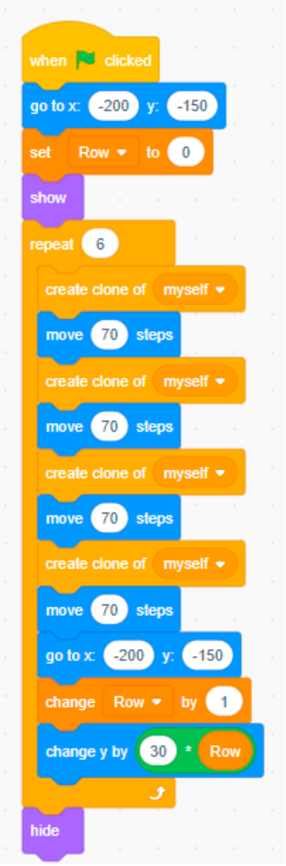
Note: This is a good opportunity to try different functions and to model the trial and error approach by applying the process of elimination or the process of systematic trials. Encourage students to save code as soon as it is working and make a copy from which different functions can be tried and the effect of changing of a block can be observed. In this way, the student can always return to their working code (the risk associated with errors is therefore very low, which encourages experimentation).

Possible answer

This code puts 8 cookies on the baking sheet (2 repeats of 4 cookies). I know that each more repetition will add 4 pastries to the baking sheet. So, I can do a repeated addition of 4 until I reach 24 pastries.

Since 4 + 4 + 4 + 4 + 4 + 4 = 24, I need to modify the loop to have 6 repetitions (6 × 4 = 24).

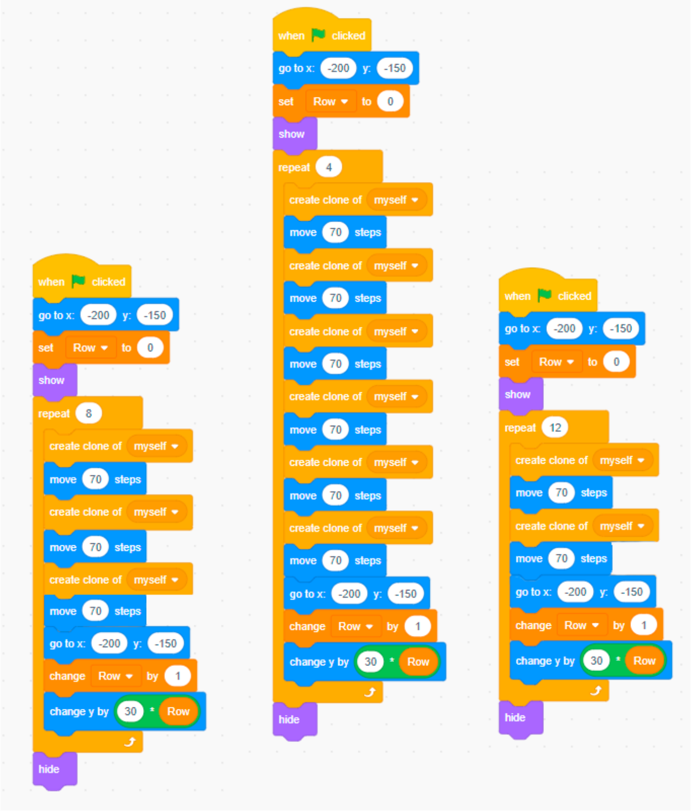
Sample code:



```
when 🏴 clicked
go to x: -200 y: -150
set Row ▾ to 0
show
repeat 6
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    go to x: -200 y: -150
    change Row ▾ by 1
    change y by (30 * Row)
↻
hide
```

Have students walk around the classroom to see other students' codes. Did the students choose to make the same alterations? Does the outcome remain the same?

Examples of possible coded solutions (always for 24 cookies):



```
when 🏴 clicked
go to x: -200 y: -150
set Row ▾ to 0
show
repeat 8
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    go to x: -200 y: -150
    change Row ▾ by 1
    change y by 30 * Row
hide
```

```
when 🏴 clicked
go to x: -200 y: -150
set Row ▾ to 0
show
repeat 4
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    go to x: -200 y: -150
    change Row ▾ by 1
    change y by 30 * Row
hide
```

```
when 🏴 clicked
go to x: -200 y: -150
set Row ▾ to 0
show
repeat 12
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    go to x: -200 y: -150
    change Row ▾ by 1
    change y by 30 * Row
hide
```

Represents 8 × 3 = 24 Represents 4 × 6 = 24 Represents 12 × 2 = 24

Examples of student thinking:

- No, the codes for other students are different. I chose to do 6 rows of 4 pastries, but I also saw 8 rows of 3 pastries and 4 rows of 6 pastries.
- There are many multiplication expressions that make 24, but I think 4 rows of 6 pastries is the best expression to use space as well.
- The outcomes are the same, there are always 24 pastries on the baking sheet.
- The outcomes are different, even though there are still 24 pastries on the baking sheet, because the layout of the pastries is different.

- Have students create a code for today's orders.
    - 20 cranberry scones
    - 81 raisin cookies
    - 47 raspberry turnovers

- Encourage students to compare their representations to those of other students.

Possible Answers

20 cranberry scones:

- I used 2 × 10, and Valerie used 4 × 5.
- I believe the 2 × 10 array is not as efficient as the 4 × 5 array, since a lot of space is wasted on the baking sheet.
- I notice that there are several ways to represent 20 cookies using the array: 4 × 5, 5 × 4, 2 × 10, and 10 × 2.

81 raisin cookies:

- There is just one way to represent this number, since there is just one multiplication that has a product of 81, which is 9 × 9.

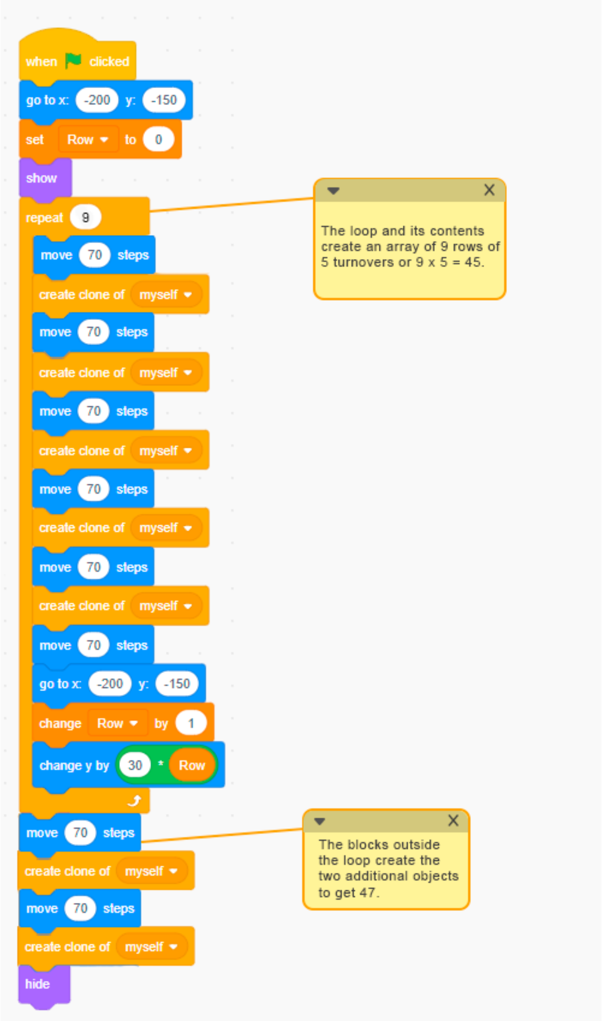
Note: Multiplication facts up to 10 × 10 are prioritized in this learning situation. However, it is possible to accept 3 × 27 or 27 × 3 answers if it occurs.

- I notice that the pastries form a square.

47 raspberry turnovers:

There are no two numbers that when multiplied equal 47, but I know that 9 × 5 = 45. First, I can create a code to put 45 turnovers on the baking sheet (9 rows of 5 turnovers) and then add two more to the side.

Sample code:

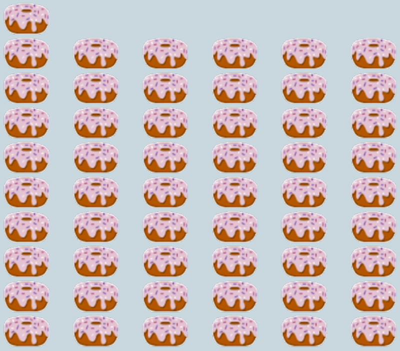


```
when ⚑ clicked
go to x: -200 y: -150
set Row ▾ to 0
show
repeat 9
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    create clone of myself ▾
    move 70 steps
    go to x: -200 y: -150
    change Row ▾ by 1
    change y by (30 * Row)
move 70 steps
create clone of myself ▾
move 70 steps
create clone of myself ▾
hide
```

The loop and its contents create an array of 9 rows of 5 turnovers or 9 x 5 = 45.

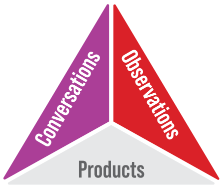
The blocks outside the loop create the two additional objects to get 47.

Note: This same reasoning could be used with other number facts, such as 6 × 7 = 42, and 5 more is 47.

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student fails to use the loop correctly.	Use different approaches to explain structure. The loop, for example, resembles a mouth, and whatever is in the mouth is chewed a certain number of times. Ask the student if the loop contains all the elements that have to be repeated (sometimes there are omissions, especially towards the end of the loop). Ask the student to name the multiplication that they are trying to represent with the code. Where can they find the two multiplied numbers in their code? Provide materials for the student to act like the sprite in an unplugged coding activity by following the code as instructed; for example, a counter = a pastry that one must put on a rectangular baking sheet. The student can then translate their actions into code for the computer.
The student does not alter the movements, so the pastries do not all fit on the baking sheet or fill it.	Ask the student to execute their code, then ask questions about their observations. Ask the student if the baking sheet was used efficiently. Ask the student if it is possible to see all the pastries on the baking sheet. Review with the student, in the context of coding, the following terminology: *advance, go to* and *add.*
The student forgets to hide the object at the end of the code. Example: 	Ask the student to count the number of pastries on the baking sheet. Ask the student to explain their reasoning for solving their problem before looking for the desired block. **Note:** There are several solutions to this problem; for example, the student could add a block at the end of the code, which returns the *sprite* to its original place, which creates the same visual outcome as "hiding" the *sprite*.
The student forgets the "create a clone of myself" block.	Ask the student if the pastries can be counted after starting their code. Ask the student to explain their reasoning for solving their problem before looking for a block. **Note:** There is also a block called "stamp" which can produce a similar outcome.

Review (15 minutes)

Assessment can be carried out through...



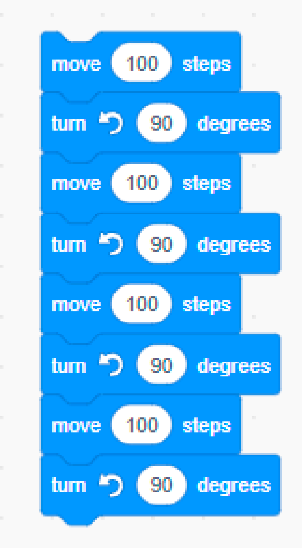
Choose a few students and ask them to present or explain their code (using pseudocode). During this discussion, ask the other students in the class to highlight similarities and differences between the codes.

Review the effective use of loops in code.

Consolidation of Learning

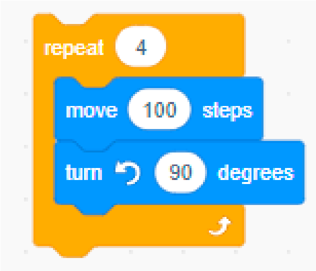
Ask the students to write codes to represent regular geometric shapes. If the coding software being used has an option to draw lines (for example, the "pen" in Scratch), students can draw shapes using *sprite* moves and rotations. Loops reduce the amount of code to achieve an outcome.

To code a square trajectory, for example, it is necessary to know the properties of the square (4 right angles and 4 congruent sides). The code for the trajectory could therefore look like this:



```
move 100 steps
turn ⟲ 90 degrees
move 100 steps
turn ⟲ 90 degrees
move 100 steps
turn ⟲ 90 degrees
move 100 steps
turn ⟲ 90 degrees
```

Adding loops can reduce the code to this:



```
repeat 4
    move 100 steps
    turn ⟲ 90 degrees
```

Students can apply this same process to create code for rectangles.

CONSIDERATIONS

Links to Other Curriculum Expectations (Repeating Events)

Number

B2.6 Represent multiplication of numbers up to 10 × 10 and division up to 100 ÷ 10, using a variety of tools and drawings, including arrays.

By using a loop, the student can make the connection between repeated addition and multiplication. By coding movements and reproductions of *sprites*, the student can progress from a representation of groups of objects towards the array model.

Algebra

C1.1 Identify and describe repeating elements and operations in a variety of patterns, including patterns found in real-life contexts.

Sequential sets of instructions inside a loop represent the pattern core of a repeating pattern. Students can therefore use pseudocode to describe the action inside the loop.

Spatial Sense

E2.8 Use appropriate non-standard units to measure area, and explain the effect that gaps and overlaps have on accuracy.

By using repeating events to ensure that a multiplication expression is represented in an array, students can relate the number of objects arranged in a rectangle to the area.

Differentiated Instruction and Universal Design for Learning

- Give the unplugged blocks to the students to replace them in order to reproduce the code.
- Give students pseudocode to use as a blueprint.
- Offer students a basic code to alter according to the number of pastries desired.
- This code also lends itself well to nested events, a somewhat more complex structure, but which can make the code easier to interpret. Students who are more comfortable with code can use this structure by inserting loops inside loops.
- Adding some parameters, for example, a maximum number of pastries that can fit on a baking sheet, which creates a more complex mathematical situation to represent using code.

Appendix – Warm-Up

Learning Situation – Grade 4

Title: In Pursuit of Patterns!
Duration: 100 minutes

Overview

In this learning situation, the student uses the power of nested control structures to create and extend repeating and growing patterns.

Overall and Spectific Expectations

Algebra

C1. Identify, describe, extend, create, and make predictions about a variety of patterns, including those found in real-life contexts.

C1.1 Identify and describe repeating and growing patterns, including patterns found in real-life contexts.

C1.2 Create and translate repeating and growing patterns using various representations, including tables of values and graphs.

C3. Solve problems and create computational representations of mathematical situations using coding concepts and skills.

C3.1 Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves sequential, concurrent, repeating and nested events.

C3.2 Read and alter existing code, including code that involves sequential, concurrent, repeating and nested events, and describe how changes to the code affect the outcomes.

Preferred High-Impact Instructional Practices in Mathematics

Math Conversations

The Warm-Up below is an opportunity to have a math conversation with the students. This opportunity allows them to exchange their ideas, show their understanding of patterns, reason and prove their reasoning as to the predicted outcome following the reading of a code. Students are able to make connections with lived experiences and share their knowledge of the different possible patterns and the various possible outcomes of a code.

Flexible Groupings

Flexible groupings while creating or altering codes facilitate communication and collaboration, among other things. Students can learn from each other and thus develop their mathematical thinking. While taking into account the profile of the students, the groupings can be random, heterogeneous or homogeneous, depending on the needs. During the course of this learning situation, flexible groupings are suggested in order to make learning optimal for the students.

Small-Group Instruction

Small-group instruction is a practice to advance student learning. Throughout the creation or alteration of a code, it is important to form small groups of students in order to provide them with the necessary instruction as needed. Small-group instruction allows students to review mathematical concepts that support their new learning. Students are therefore able to reinforce their understanding of the concepts under study.

Direct Instruction

When a code does not give the expected outcome, teachers could ask students to explain it. This is a good time to offer direct instruction related to the difficulty observed. This instruction verifies students' understanding and provides immediate feedback.

Prior Knowledge and Skills

- The Grade 4 student has seen in previous grades how to create code using block-based coding software. If not, an introduction to coding software and exploration time is required for this activity to work well.
- Several coding software programs use the Cartesian plane to determine the position and movements of objects. It would be important to see the orientation of the x and y axes; for example, if one adds to the value of x, the object moves forward, while, if one adds to the value of y, the object ascends. A presentation of this concept could be useful.

Learning Goals

At the end of this learning situation, the student will be able to:

- use the terminology under study such as *code, sprite, sequential events, concurrent events, repeating events, nested events, repeating pattern, growing pattern, table of values;*
- describe patterns;
- create repeating and growing patterns by writing and executing codes including sequential events, concurrent events, repeating events, and nested events;
- read and alter codes including sequential events, concurrent events, repeating events and nested events.

Criteria According to the Achievement Chart

Knowledge and Understanding

- The student knows the blocks of block-based programming software and understands how they work.
- The student recognizes the pattern core in a repeating pattern (with manipulatives or in a code).

Thinking

- The student plans to write or alter code by writing representative pseudocode.
- The student predicts the outcome of a code.
- The student organizes a representation of a growing pattern in a table.
- The student alters a code to incorporate a nested structure (a loop within a loop) in order to make the code more relevant to a specific mathematical situation.

Communication

- The student makes appropriate use of vocabulary related to coding *(code, blocks, sprite, character, sequential events, concurrent events, repeating events, nested events)* and patterns *(repeating patterns, growing patterns, position, pattern rule)*.
- The student writes code and pseudocode respecting the format and the syntax (for example, the use of indents to demarcate what is in a loop).

Application

- The student alters their code so that it represents a new situation similar to the original problem.
- Students use their knowledge of repeatng patterns to determine the pattern rule of a pattern created with code.

Materials

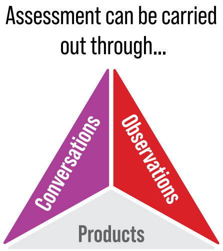
- computer equipped with appropriate block-based coding software according to the level of the students
- objects representing patterns (two-coloured tokens, coloured tiles, etc.)
- appendix (Warm-Up)

Note: The examples in this learning situation were created with Scratch.

Mathematical Vocabulary

coding-specific vocabulary *(code, sprite, sequential set of instructions, loop)*, sequential events, concurrent events, repeating events, nested events, repeating and growing patterns, pattern, term, position, table of values

Before Learning (Warm-Up) (15-20 minutes)



Show the image below (appendix Warm-Up) to the class and lead a "I notice, I wonder" type discussion. Take note of the students' comments and questions.

Display the table of values below on the screen and ask the students if adding the table affects the comments and questions shared when only seeing the image. Add new comments and questions to the list.

MONTH	MONEY ADDED TO PIGGY BANK	SUM OF MONEY IN THE PIGGY BANK
January	$5	$5
February	$5	$10
March	$5	$15
June	$5	?

Examples of student thinking:

- I notice that we always add the same amount to the piggy bank.
- I notice that we jump directly from March to June. Months are missing.
- I notice that the amount of money increases over time.
- I wonder how much money will be in the piggy bank in the month of June.
- I wonder if the amount of money will stay the same throughout the year.
- I wonder if the person will spend money from their piggy bank before June.

Provide students with manipulatives and have them represent the data to determine the amount of money in the piggy bank in the month of June.

Example of possible representation with counters:

● = $1

● ● ● ● ●	● ● ● ● ● ● ● ● ● ●	● ● ● ● ● ● ● ● ● ● ● ● ● ● ●	● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●	● ●	● ●
1	2	3	4	5	6
January	February	March	April	May	June

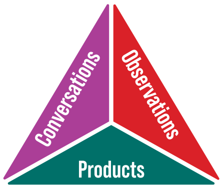
Check to see if the students' predictions of the amount of money in the piggy bank were correct. If students got different results, check to see if this is because of assumptions that were made (for example, one month the person gets more than $5, or one month the person decides to incur an expense).

Explain to students that representation with manipulatives works relatively well in this situation. But, if one wanted to determine the savings over a longer period, it could take a long time to represent. To make this process faster and more user-friendly, we can write code that generalizes the situation and allow us to see the savings further into the future.

Note: It is not so much a question here of the "answer", but rather of the representation of the mathematical situation. Students may quickly notice that you can multiply the number of months (the position, or term number) by 5 to get the total amount of money saved. However, this does not show us the growth of the amount of money in the piggy bank.

Active Learning (Exploration)
(60 minutes, in addition to the time allocated to explore the software, if necessary)

Assessment can be carried out through...

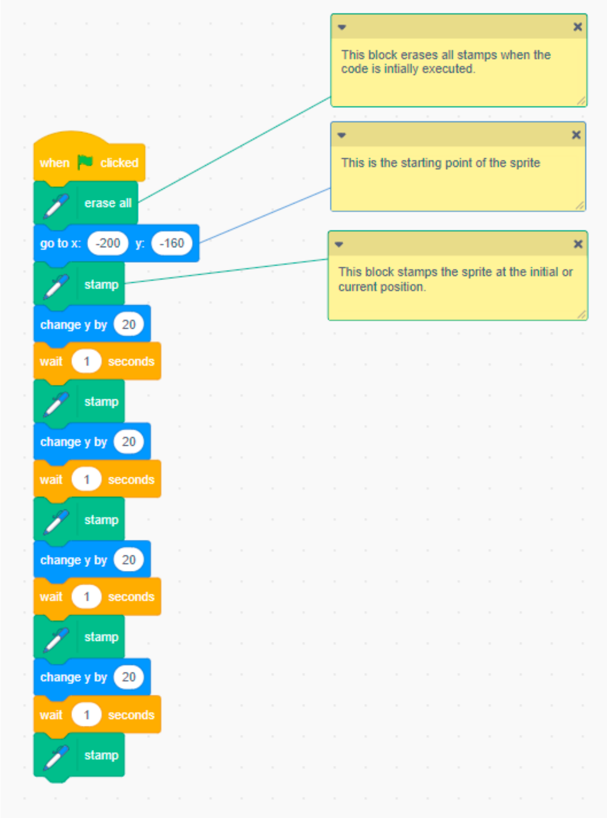
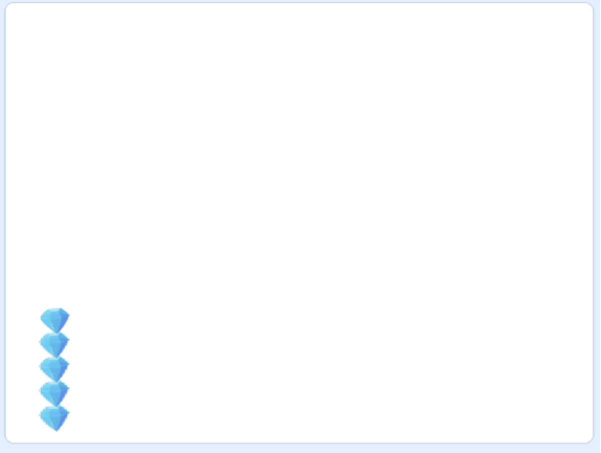


Present students with a code that represents the piggy bank situation. Do not show them the result of the code execution (do not click on the green flag).

Note: In the examples provided, the yellow notes are explanations of how the code works and should be omitted when the code is presented to students.

Have students study the code to predict its outcome. Writing a pseudocode could also help the student with their prediction and better understand what the various blocks are used for.

Sample code representing the piggy bank problem using counters:

CODE	OUTCOME
	

```
when 🏁 clicked
erase all
go to x: -200 y: -160
stamp
change y by 20
wait 1 seconds
stamp
change y by 20
wait 1 seconds
stamp
change y by 20
wait 1 seconds
stamp
change y by 20
wait 1 seconds
stamp
```

This block erases all stamps when the code is intially executed.

This is the starting point of the sprite

This block stamps the sprite at the initial or current position.

Example of a pseudocode representative of the code presented:

```
Starting Condition – Green Flag
    Object: counter that goes to the lower left corner
        1 second pause
            The position of the counter is stamped.
    Object: counter that moves upwards
        1 second pause
            The position of the counter is stamped.
    Object: counter that moves upwards
        1 second pause
            The position of the counter is stamped.
    Object: counter that moves upwards
        1 second pause
            The position of the counter is stamped.
    Object: counter that moves upwards
        1 second pause
            The position of the counter is stamped.
```

After allowing enough time for the students to make their predictions, the code can be executed. Then, the students can compare their predictions with the result obtained.

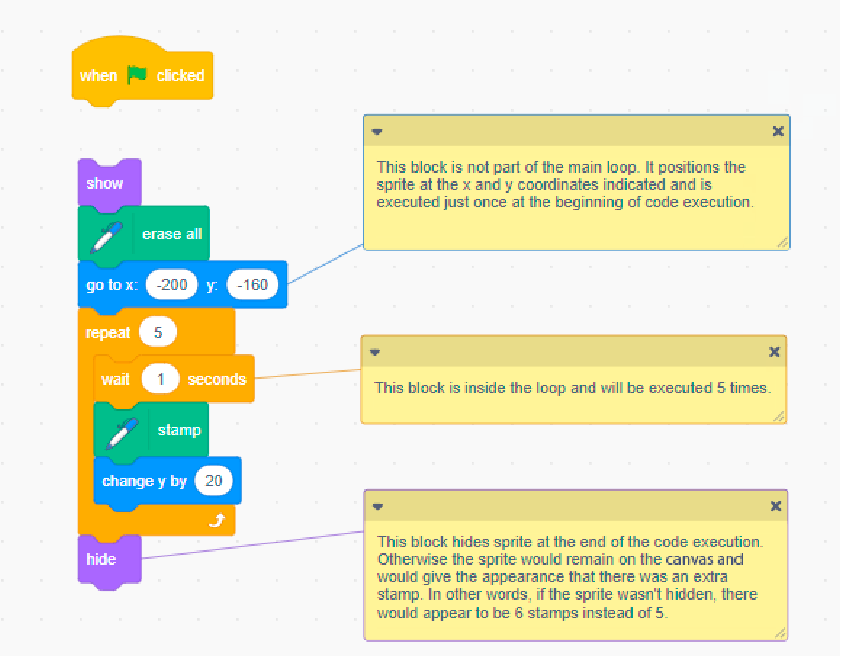
Ask students what stands out most in the code just explored. There are several possible answers, but try to direct the questioning to the elements that repeat. The code, as presented, is a repeating pattern. The pattern core is as follows:

(Counter move), (1 second pause), (Stamp)

This pattern core repeats throughout the code. This can therefore be reduced with a loop.

Note: If students are unfamiliar with the concept of a loop, it would be beneficial to have them explore coding software using the loop so that students become familiar with this structure before continuing.

Example of reduced code with the loop:



```
when 🏁 clicked

show
erase all
go to x: -200 y: -160
repeat 5
    wait 1 seconds
    stamp
    change y by 20
hide
```

This block is not part of the main loop. It positions the sprite at the x and y coordinates indicated and is executed just once at the beginning of code execution.

This block is inside the loop and will be executed 5 times.

This block hides sprite at the end of the code execution. Otherwise the sprite would remain on the canvas and would give the appearance that there was an extra stamp. In other words, if the sprite wasn't hidden, there would appear to be 6 stamps instead of 5.

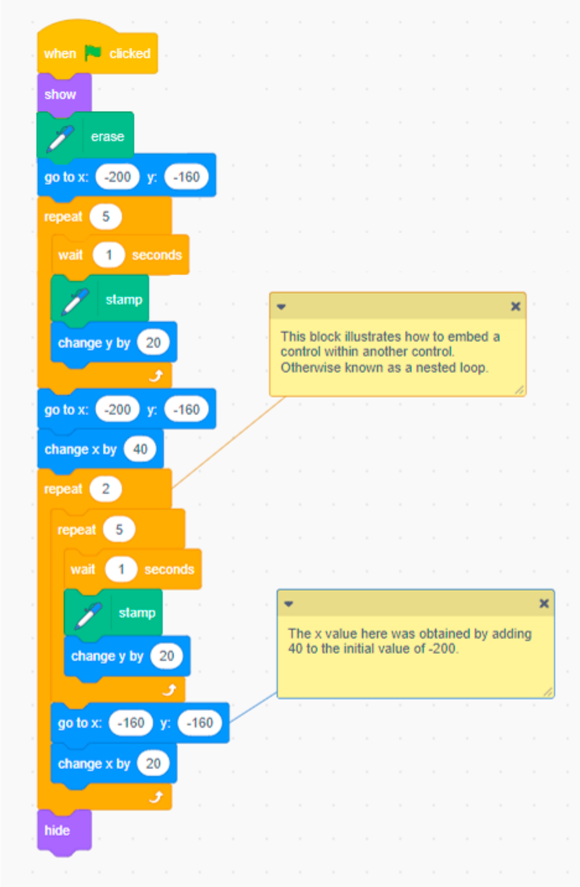
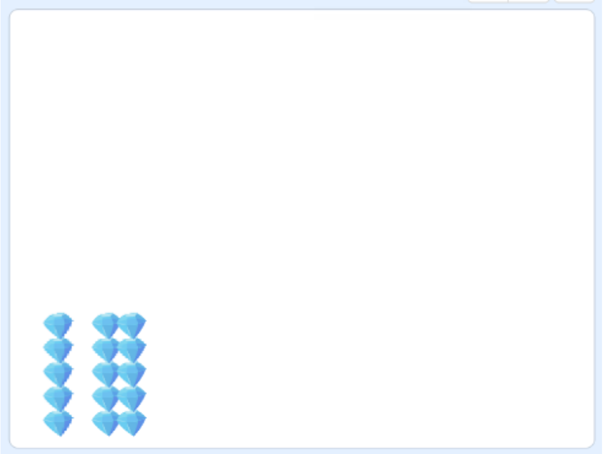
Note that the outcome of this code gives the same outcome as the longer code without the loop. Point out to the students that the code also produces a representation identical to the term in position 1 – January, created during the Warm-Up. Ask them to alter the code so that it reproduces the mathematical situation of the piggy bank. There are several ways to proceed.

1. If students are beginning their exploration of coding, it would be beneficial to show them, through trial and error, how to alter code to insert loops. By modelling their suggestions, students can work together to find a solution to the problem.
2. If students have a little more experience with coding software, pairs or even individual work would be beneficial for students to discover code that will represent the mathematical situation.
3. If the level of experience and comfort of the students varies a lot in the class, random or strategic groupings could put them in a peer learning situation. Teachers could also take this opportunity to do direct instruction in small groups according to the needs of the students.

Here is an example of pseudocode and code that represents the piggy bank situation with loops:

```
Starting Condition – Green Flag
    Sprite: counter that goes to the lower left corner (initial position 1)
        Repeat 5 times.
            1 second pause
            The position of the token is stamped.
            Sprite: counter that moves up 20 units
    Sprite: counter that returns to the initial position 1
    Sprite: counter that moves to the right by 30 units (initial position 2)
        Repeat 2 times.
            Repeat 5 times.
                1 second pause
                The position of the counter is stamped.
                Sprite: counter that moves up 20 units
    Sprite: counter that returns to the initial position 2
    Sprite: counter that moves to the right by 2 units
```

This pseudocode represents 2 months (initial position 1 and initial position 2). By determining the trend, we can extend this code, which again looks like a pattern, to represent several months.

CODE	OUTCOME
	

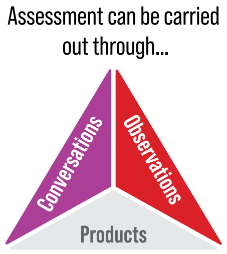
```
when ⚑ clicked
show
erase
go to x: -200 y: -160
repeat 5
    wait 1 seconds
    stamp
    change y by 20
go to x: -200 y: -160
change x by 40
repeat 2
    repeat 5
        wait 1 seconds
        stamp
        change y by 20
    go to x: -160 y: -160
    change x by 20
hide
```

This block illustrates how to embed a control within another control. Otherwise known as a nested loop.

The x value here was obtained by adding 40 to the initial value of -200.

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student does not understand how the loop works.	Show the student a simpler code example with a loop. Ask the student to write code with a single event in a loop and observe the outcome.
Student does not understand how a loop within a loop works.	Execute each loop separately so the student can see its outcome without it being nested. Remind the student of the multiplicative effect of nested loops to provide a more accessible benchmark.
The student does not recognize the pattern rule (row 1 = one column of 5, row 2 = 2 columns of 5, etc.).	Give the student manipulatives to represent the situation and identify the pattern rule. Ask the student to verbalize their observations regarding the pattern rule. Sometimes it is through conversation that understanding occurs.
The student code bears no resemblance to the written pseudocode.	Coding software often uses colour codes for blocks. Ask the student to add colour, using highlighters, to their pseudocode, respecting the colours used in the software in question. The pseudocode and code can then be compared to find inconsistencies.
The student code does not work.	Ask the student if it is a syntax error (the code does not work at all) or a logical error (the code works, but does not produce the desired outcome). Create code verification partnerships between students in the class. Sometimes it's easier to find an error if you haven't created the code. Ask the student to explain the role of each code, especially nested events, with pseudocode.

Review (20 minutes)



Ask students about the use of the loop within the loop (nested events).

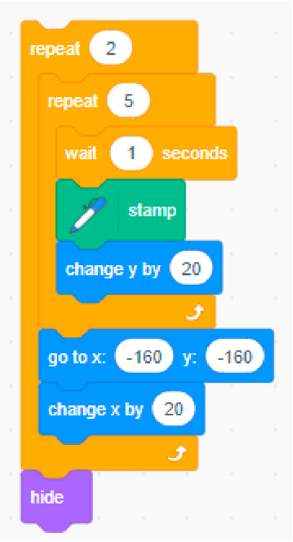
- What parts of the visual are the outcome of the inner loop?

  The five objects in each column.

- What parts of the visual are the outcome of the outer loop?

  The number of object columns per term.

- What would happen if we reversed the numbers of repeats in the inner and outer loop of the next code?



```
repeat 2
    repeat 5
        wait 1 seconds
        stamp
        change y by 20
    go to x: -160 y: -160
    change x by 20
hide
```

Instead of two columns of five objects, we would have five columns of two objects.

Ask students if this represents income, savings, or investment. Answers may vary depending on the interpretation of the problem. A discussion of terminology would be helpful if these terms have not already been seen in class.

Possible answers

- It is a monthly income, because the person was paid $5 per month, the sum added to the piggy bank.
- These are savings because the person chooses to set aside a certain amount of money each month.
- It is not an investment, the only thing that affects the value of the piggy bank is the person's deposits.

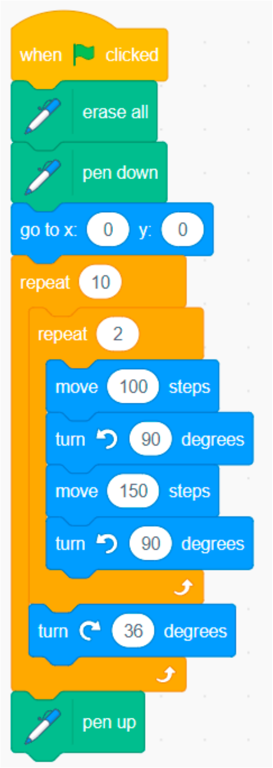
Consolidation of Learning

- Provide students with different situations to solve using nested events.

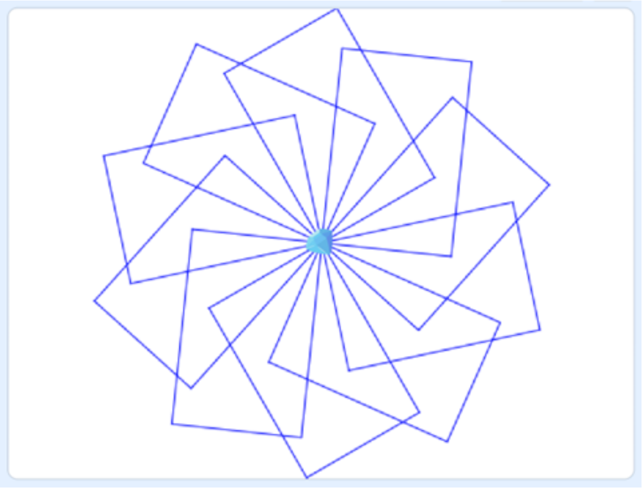
Examples:

- The person makes the same payments as in the initial situation (regular payments of $5 per month), but spends $10 every three months.
- Instead, the person deposits an additional $1 per month ($5, $6, $7, $8, etc.).
- The person doubles the value of their piggy bank each month (total amount in the piggy bank: $5, $10, $20, $40).

- Invite students to return to the coding software to represent a new situation of their choice based on their existing code.
- Have students create geometric patterns using nested loops. Using the properties of a rectangle, students can create a code that draws a rectangle, then rotates it slightly before drawing another one, a number of times. The resulting code and pattern might look like this:



```
when 🏁 clicked
erase all
pen down
go to x: 0 y: 0
repeat 10
    repeat 2
        move 100 steps
        turn ↺ 90 degrees
        move 150 steps
        turn ↺ 90 degrees
    turn ↻ 36 degrees
pen up
```



- Have students experiment with changing the number of repetitions of the outer loop and the degrees of rotation with each repetition to create new patterns.

CONSIDERATIONS

Links to Other Curriculum Expectations

Number

B2.2 Recall and demonstrate multiplication facts for 1 × 1 to 10 × 10, and related division facts.

Nested loops create more complex multiplication representations than simple loops. It is possible, for example, to obtain 24 repetitions of a code by placing it in a loop of 4 repetitions, and placing the loop of 4 repetitions in a loop of 6 repetitions.

Algebra

C1.1 Identify and describe repeating and growing patterns, including patterns found in real-life contexts.

C1.2 Create and translate repeating and growing patterns using various representations, including tables of values and graphs.

C2.1 Identify and use symbols as variables in expressions and equations.

Nested events create growing patterns. A first loop defines the pattern rule, and it is placed in a second loop which determines the number of repetitions. Thereafter, it is a question of adding code blocks and variables to alter the pattern rule.

Spatial Sense

E2.5 Use the row and column structure of an array to measure the areas of rectangles and to show that the area of any rectangle can be found by multiplying its side lengths.

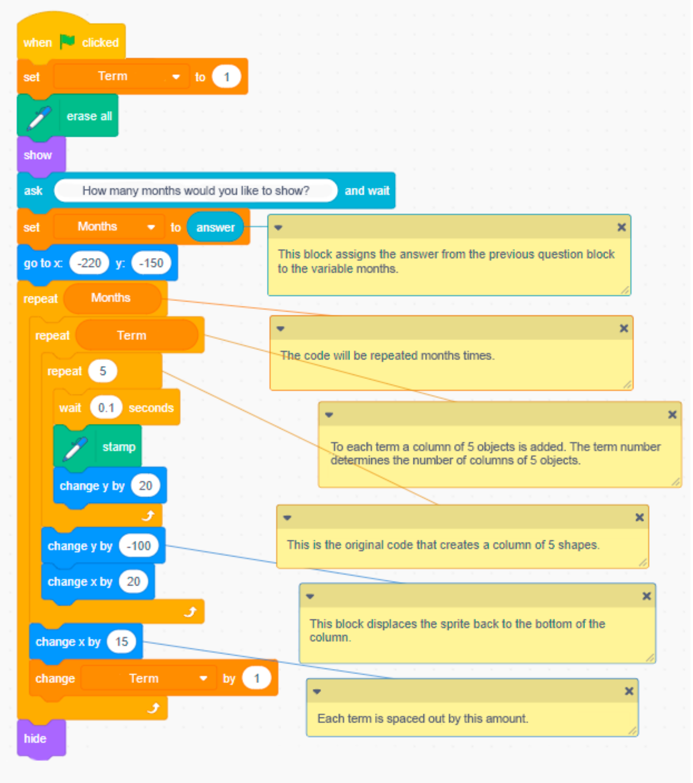
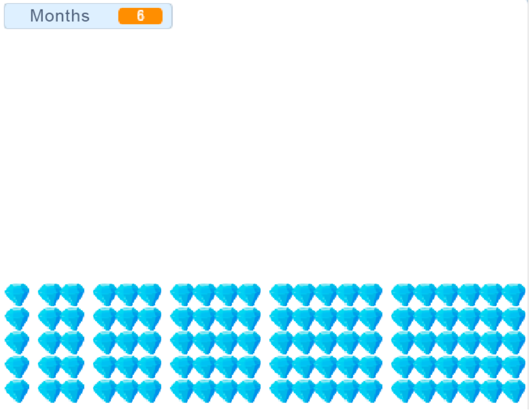
Nested loops can help represent the array. The inner loop represents the number of elements in a row, and the outer loop represents the number of rows.

Financial Literacy

F1.3 Explain the concepts of spending, saving, earning, investing, and donating, and identify key factors to consider when making basic decisions related to each.

Differentiated Instruction and Universal Design for Learning

Students comfortable with block-based coding could use variables to further automate the code. See below for example code that asks for the number of months to represent. The student uses the answer to represent the mathematical situation. The code could also be studied in order to predict the outcome or provided for the student to alter according to their personal objectives.

CODE	OUTCOME
	

```
when [flag] clicked
set Term to 1
erase all
show
ask How many months would you like to show? and wait
set Months to answer
go to x: -220 y: -150
repeat Months
    repeat Term
        repeat 5
            wait 0.1 seconds
            stamp
            change y by 20
        change y by -100
        change x by 20
    change x by 15
    change Term by 1
hide
```

This block assigns the answer from the previous question block to the variable months.

The code will be repeated months times.

To each term a column of 5 objects is added. The term number determines the number of columns of 5 objects.

This is the original code that creates a column of 5 shapes.

This block displaces the sprite back to the bottom of the column.

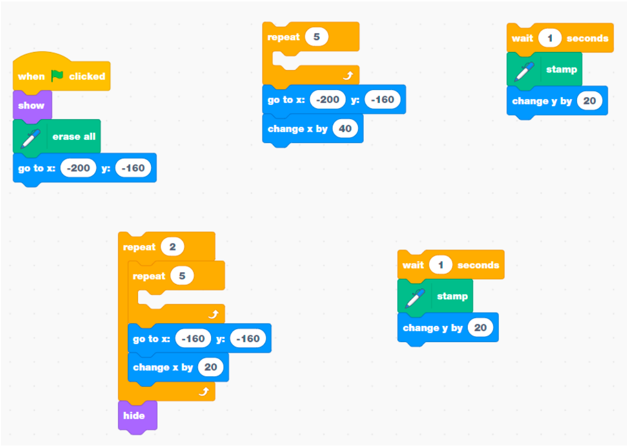
Each term is spaced out by this amount.

The student who has difficulty using the coding software could team up with students who are more comfortable in order to learn from their partners. A group of students with the same degree of experience with the software would also facilitate direct instruction according to the needs of the group.

Several coding software allow the sharing of projects. Students could be given certain assembled code blocks in advance in order to put them in the right places to make the code work.

Example:

THE STUDENT STARTS WITH THIS...



```
when 🏁 clicked
show
erase all
go to x: -200 y: -160
```

```
repeat 5

go to x: -200 y: -160
change x by 40
```

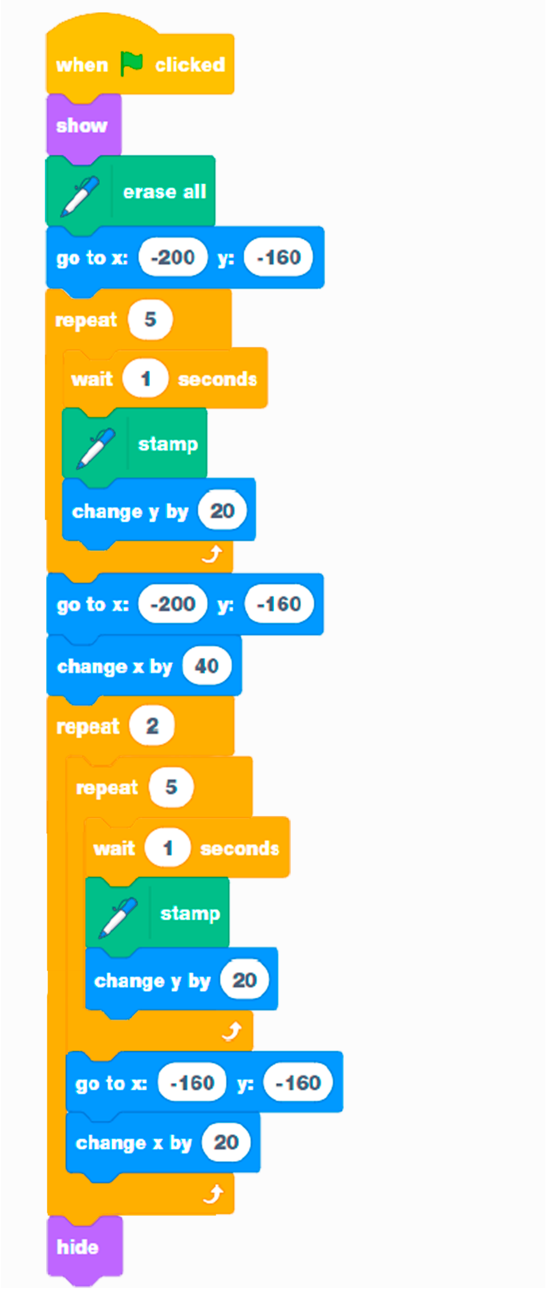
```
wait 1 seconds
stamp
change y by 20
```

```
repeat 2
    repeat 5

    go to x: -160 y: -160
    change x by 20
hide
```

```
wait 1 seconds
stamp
change y by 20
```

... IN ORDER TO CREATE THIS



```
when 🏁 clicked
show
erase all
go to x: -200 y: -160
repeat 5
    wait 1 seconds
    stamp
    change y by 20
go to x: -200 y: -160
change x by 40
repeat 2
    repeat 5
        wait 1 seconds
        stamp
        change y by 20
    go to x: -160 y: -160
    change x by 20
hide
```

Appendix – Warm-Up

MONTH	MONEY ADDED TO PIGGY BANK	SUM OF MONEY IN THE PIGGY BANK
January	$5	$5
February	$5	$10
March	$5	$15
June	$5	?

Learning Situation – Grade 5

Title: The Probability Machine
Duration: 125 minutes

Overview

In this learning situation, the student creates a machine to use the random roll of a die in order to show that the higher the number of trials, the closer the experimental probability approaches the theoretical probability.

Overall and Specific Expectations

Algebra

C3. Solve problems and create computational representations of mathematical situations using coding concepts and skills.

C3.1 Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves conditional statements and other control structures.

C3.2 Read and alter existing code, including code that involves conditional statements and other control structures, and describe how changes to the code affect the outcomes.

Data

D2. Describe the likelihood that events will happen, and use that information to make predictions.

D2.2 Determine and compare the theoretical and experimental probabilities of an event happening.

Preferred High-Impact Instructional Practices in Mathematics

Learning Goals, Success Criteria and Descriptive Feedback

Post the success criteria in the classroom for students to refer to as they work. Throughout the lesson, take the time to circulate and observe what the students are doing. Take the opportunity to question them about their work and give them descriptive feedback on the success criteria displayed.

Math Conversations

During Warm-Up, the students discuss the possible probabilities by rolling the die. As they explore possible outcomes, students listen to different opinions and explain their reasoning to their peers. Take the time to circulate and question the students about their observations, their knowledge or their new learning goals.

Flexible Groupings

Students work in groups to foster collaboration. Small group experiences provide students with confidence and often motivate them to take risks. Students can work in small groups to create their probability machine that facilitates, in time and accuracy, the 1000 required throws.

Prior Knowledge and Skills

To be able to carry out this learning situation, the student must:

- have an understanding of experimental probability and theoretical probability;
- be able to organize data into a relative-frequency table (fractions);
- be able to use block-based programming software;
- know the multiples of numbers.

Learning Goals

At the end of this learning situation, the student will be able to:

- write and execute code for a random die roll in block-based programming software;
- integrate one or more conditional statements into their code;
- compare the theoretical and experimental probabilities of an event.

Criteria According to the Achievement Chart

Knowledge and Understanding

- The student understands the meaning of conditional statement blocks.
- The student knows the difference between theoretical probability and experimental probability.
- The student determines the blocks and actions of each block used in repeating and nested events.
- The student uses coding software to represent a random result.

Thinking

- The student writes pseudocode accurately.
- The student compares the theoretical and experimental probabilities of obtaining an even number.
- The student alters/debugs their code with precision when it does not work.

Communication

- The student clearly explains the impact of certain blocks such as control blocks.
- The student uses vocabulary related to coding *(conditional statements, variables)* and probabilities *(impossible, unlikely, equally likely, likely, certain)*.

Application

- The student creates an efficient code to count the random throws of a die.
- The student creates an efficient code to count the randomly drawn even numbers.

Materials

- block-based programming software
- die numbered from 1 to 6
- pencil
- appendix (Dice)

Note: The examples in this learning situation were designed with Scratch.

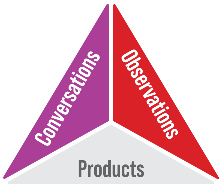
Mathematical Vocabulary

experimental probability, theoretical probability, repeating events, nested events, conditional statement, control structure

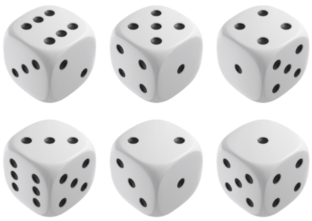
CONTENTS

Before Learning (Warm-Up) (10 minutes)

Assessment can be carried out through...



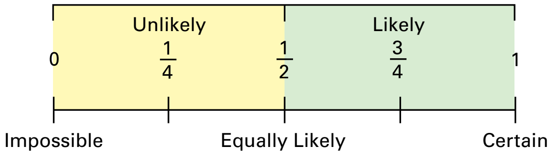
Show the image of the dice (appendix Dice) to the class. Ask students to come up with questions to ask themselves about this picture.



Sample questions:

- How many dots are there altogether?
- How often do we see the number 6?
- What is the probability of rolling a 3?

Group the students into small groups and give each one a die numbered 1 to 6. Ask them to find the theoretical probability of rolling an even number. If necessary, review the probability line and associated terminology.



Example of sought-after reflection:

- There are three even numbers: 2, 4 and 6. There are six possible outcomes when rolling a die. The probability of having an even number is therefore $\frac{3}{6}$.
- The fraction $\frac{3}{6}$ can be simplified to $\frac{1}{2}$, which is also half. On the probability line, this tells us that it is equally likely to obtain an even number or an odd number.

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student does not know where to start.	What should you do first? What are all the possible outcomes of a die roll? What are the even numbers on the die? What is a probability?
The student has difficulty expressing probability as a fraction.	What does the numerator represent in probability? What does the denominator represent in probability? How many sides does the die have in total? Is this total the numerator or the denominator?
The student has difficulty expressing the probability of rolling an even number as a fraction.	What does the fraction $\frac{3}{3}$ represent? (a whole number, $\frac{1}{1}$ or 100%) Is it true that each roll of the die will result in an even number?

Ask students to find the theoretical probability of rolling a multiple of 3, then the probability of rolling the number 4. Students should then place the probabilities on the probability line, as in the first example.

Example of sought-after reflection:

- There are two multiples of 3 on a six-sided die (3 and 6), and six possible outcomes, so the probability of rolling a multiple of 3 is $\frac{2}{6}$.
- The fraction $\frac{2}{6}$ is equivalent to $\frac{1}{3}$.

or

- I know that $\frac{1}{3}$ is between $\frac{1}{4}$ and $\frac{1}{2}$.
- There is only one number 4 on a six-sided die, and six possible outcomes, so the probability of rolling a 4 is $\frac{1}{6}$.

Ask students to roll the die 20 times and compile the results into a relative-frequency table like this one.

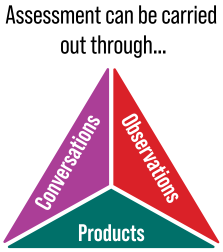
EVENT	COUNT	FREQUENCY	RELATIVE FREQUENCY EXPRESSED AS A FRACTION
I get an even number.			
I get a multiple of 3.			
I get a 4.			

Ask students what they notice about the results.

Do the results represent the theoretical probability? Why? (Write student responses to this question for future reference.)

Discuss the results obtained and compare them with the probabilities found earlier. Encourage students to question the effect of the number of throws on the outcome.

Active Learning (Exploration) (100 minutes)



Ask students to determine the experimental probability of rolling an even number by rolling the die 1000 times.

Example of sought-after reflection:

- 1000 rolls of the dice is a lot, and it will take a long time.
- A dice roll takes about 5 seconds, so 1000 rolls is approximately 5000 seconds (5000 seconds is about 83 minutes!).
- With so many throws, one could make more mistakes, for example forgetting to put a result in the relative-frequency table or losing count of the number of throws.

Ask the students if there is a more efficient way to determine the results of 1000 throws (we are trying to consider code as a representational tool).

First ask students what the computer should do to count the number of times an even number is rolled after 1000 rolls of a six-sided die. This becomes the pseudocode that will direct the block coding.

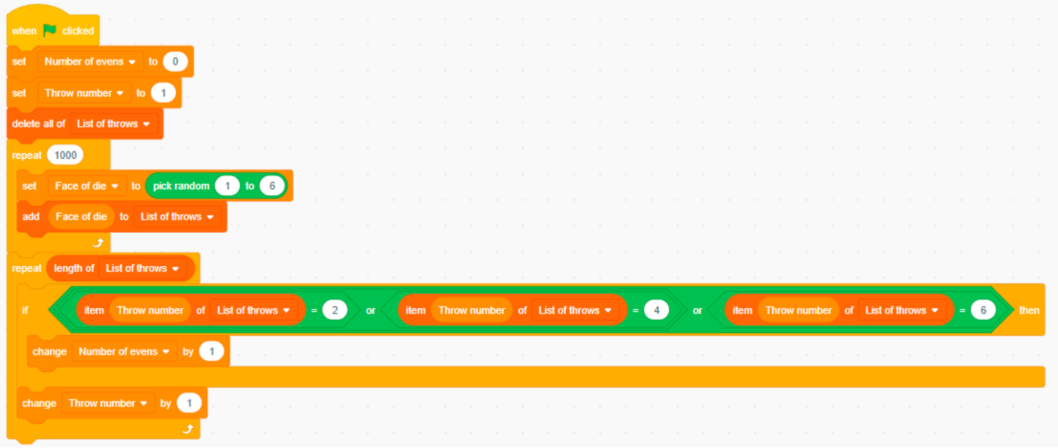
Pseudocode example:

```
Repeat 1000 times.
    Generate a random number from 1 to 6.
    Add the number to the "Throw List" list
        If number = 2 or number = 4 or number = 6,
        add 1 to the "Even Numbers" counter.
```

The code consists of three parts:

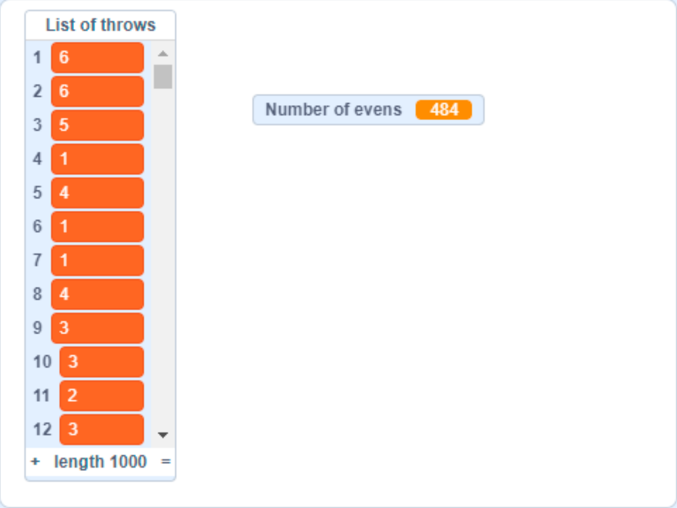
- A code for rolling a six-sided die and compiling the results.
- A code with a conditional event to determine if the result of the throw is even or not.
- A code that compiles the total of throws resulting in an even number.

Here is an example of code that determines the number of times an even number is rolled after 1000 rolls of the die, based on the pseudocode above.



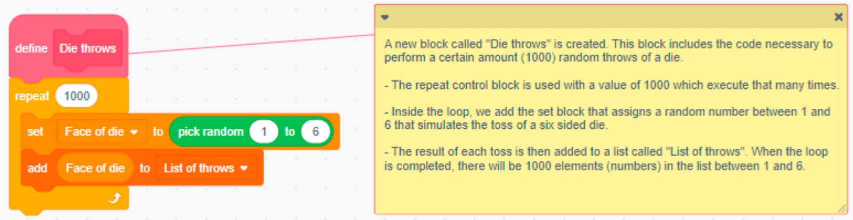
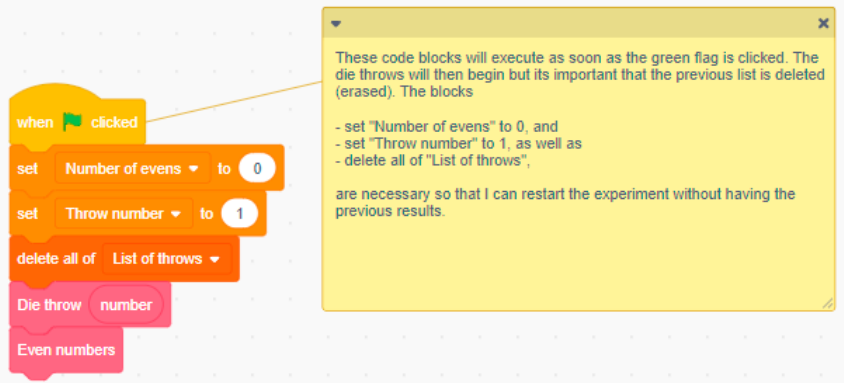
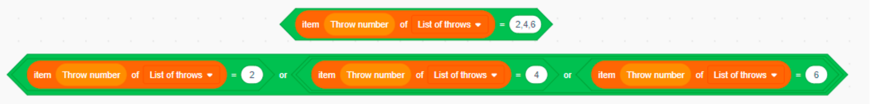
```
when ⚑ clicked
set Number of evens ▾ to 0
set Throw number ▾ to 1
delete all of List of throws ▾
repeat 1000
    set Face of die ▾ to pick random 1 to 6
    add Face of die to List of throws ▾
    ↺
repeat length of List of throws ▾
    if item Throw number of List of throws ▾ = 2 or item Throw number of List of throws ▾ = 4 or item Throw number of List of throws ▾ = 6 then
        change Number of evens ▾ by 1
    change Throw number ▾ by 1
    ↺
```

Here is the outcome generated by this code. The student code could be different while remaining functional.

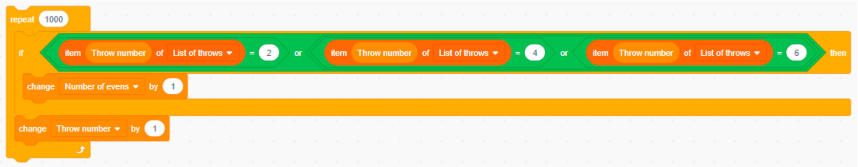
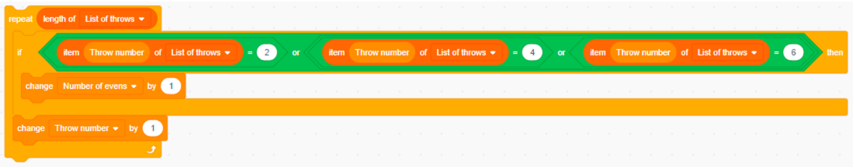


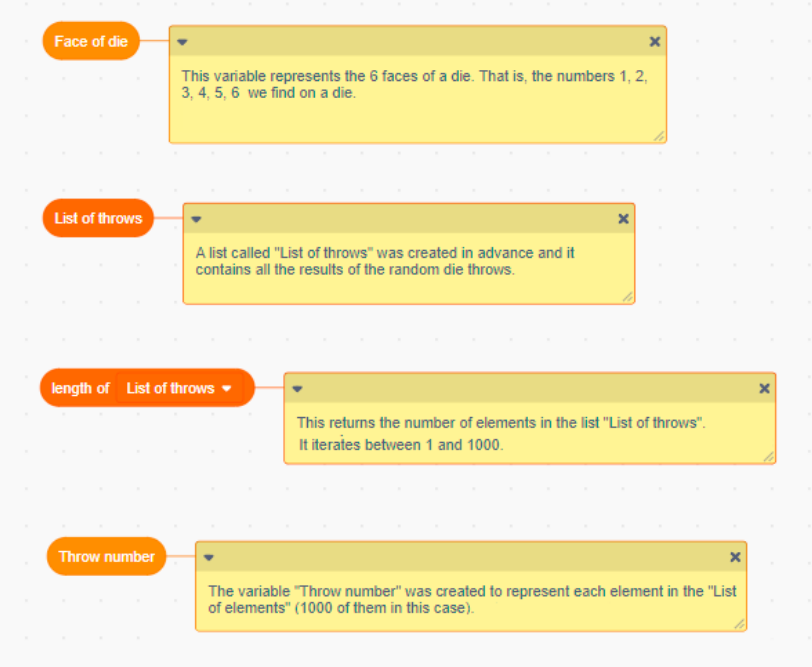
```
List of throws
1   6
2   6
3   5
4   1
5   4
6   1
7   1
8   4
9   3
10  3
11  2
12  3
+   length 1000   =

Number of evens   484
```

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student fails to compile the results of the die roll.	Ask the student to find the number of faces of the die given to them. • Which block allows you to represent the die with its six faces? • How many throws do you have to make? • Which block allows you to make multiple throws?  define Die throws repeat 1000 set Face of die to pick random 1 to 6 add Face of die to List of throws A new block called "Die throws" is created. This block includes the code necessary to perform a certain amount (1000) random throws of a die. - The repeat control block is used with a value of 1000 which execute that many times. - Inside the loop, we add the set block that assigns a random number between 1 and 6 that simulates the toss of a six sided die. - The result of each toss is then added to a list called "List of throws". When the loop is completed, there will be 1000 elements (numbers) in the list between 1 and 6.
The student forgets to reset their variables and lists to zero.	 when ⚑ clicked set Number of evens to 0 set Throw number to 1 delete all of List of throws Die throw number Even numbers These code blocks will execute as soon as the green flag is clicked. The die throws will then begin but its important that the previous list is deleted (erased). The blocks - set "Number of evens" to 0, and - set "Throw number" to 1, as well as - delete all of "List of throws", are necessary so that I can restart the experiment without having the previous results. Ask the student to read their code aloud to see if there are any instructions missing. Ask the student to explain the three blocks identified. Ask the student what happens when the variables are not reset. What is the impact on the outcomes?
The student does not use operator blocks correctly.	 item Throw number of List of throws = 2,4,6 item Throw number of List of throws = 2 or item Throw number of List of throws = 4 or item Throw number of List of throws = 6 Ask the student if there is a way to make the computer do more of what is wanted (use the "or" block). Make them understand that the computer reads one piece of data at a time, so the comma is not useful in programming. Using the "or" operator block is useful for suggesting multiple possible answers, such as 2 or 4 or 6 for even numbers.

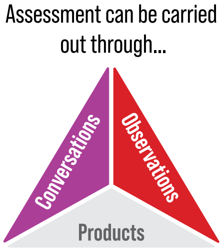
POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS

The student repeats 1000 times in their even number code rather than repeating the length of the list.	Ask the student where the results of the die rolls are compiled (the results of the die rolls should be compiled in the "Throw List").
	In which list do you have to find the even numbers?
	What variable should you add to find even numbers? (I need to add the "length of the throw list", so the whole throw list is going to be repeated.)
	
	`repeat (1000)`
	`    if <<(item (Throw number) of [List of throws ▾]) = (2)> or <<(item (Throw number) of [List of throws ▾]) = (4)> or <(item (Throw number) of [List of throws ▾]) = (6)>>> then`
	`        change [Number of evens ▾] by (1)`
	`    change [Throw number ▾] by (1)`
	
	`repeat (length of [List of throws ▾])`
	`    if <<(item (Throw number) of [List of throws ▾]) = (2)> or <<(item (Throw number) of [List of throws ▾]) = (4)> or <(item (Throw number) of [List of throws ▾]) = (6)>>> then`
	`        change [Number of evens ▾] by (1)`
	`    change [Throw number ▾] by (1)`

Student does not use variables appropriately.	Ask the student what the variables mean. The meaning of the variables used must be known in order to fully understand the code.  Face of die This variable represents the 6 faces of a die. That is, the numbers 1, 2, 3, 4, 5, 6 we find on a die. List of throws A list called "List of throws" was created in advance and it contains all the results of the random die throws. length of List of throws This returns the number of elements in the list "List of throws". It iterates between 1 and 1000. Throw number The variable "Throw number" was created to represent each element in the "List of elements" (1000 of them in this case).

Note: In the examples provided, the yellow notes are explanations of how the code works and should be omitted when the code is presented to students.

Review (15 minutes)



Review the activity.

- What are the findings?
- Do the results of the 20-throw experimental probability differ from the results of the 1000-throw experimental probability?
- What do you notice between the theoretical probability and the experimental probability?
- In your code, which blocks represent conditional statements?
- Could you use your randomizer or a similar machine in other probability situations? Which ones? Why?

Consolidation of Learning

- Suggest that the students alter their code to check the experimental probability of obtaining a multiple of 3 or the number 4.
- Suggest that the students make a code to compare the theoretical probability and the experimental probability of drawing a red marble from a bag of 10 marbles knowing that 3 marbles are red (we are dealing with the conditional *if/then/else* instructions in this example).

CONSIDERATIONS

Links to Other Curriculum Expectations

Number

B1.4 Compare and order fractions from halves to twelfths, including improper fractions and mixed numbers, in various contexts.

B1.5 Read, represent, compare, and order decimal numbers up to hundredths, in various contexts.

Conditional statements (*if/then/else*) allow you to make comparisons between numbers by using them in combination with the operator blocks (>, <, =).

Data

D2.1 Use fractions to express the probability of events happening, represent this probability on a probability line, and use it to make predictions and informed decisions.

D2.2 Determine and compare the theoretical and experimental probabilities of an event happening.

Conditional statements allow coding in experimental situations with probabilities. The *if/then/else* terminology is useful in determining certain events of a random experiment.

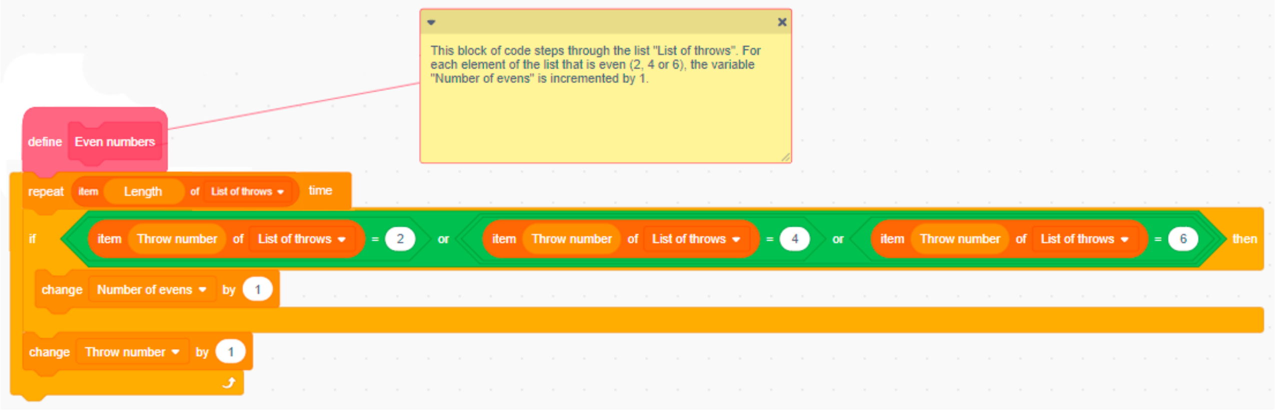
Financial Literacy

F1.2 Estimate and calculate the cost of transactions involving multiple items priced in dollars and cents, including sales tax, using various strategies.

With the elimination of the 1 cent coin, the total cost is rounded to the nearest 5 cents. Using conditional statements (*if/then/else*), it is possible to create rules to follow for rounding (for example, IF the number in the hundredths place is 1 or 2, THEN change the value to 0. IF the number in the hundredths place is 3 or 4, THEN change the value to 5, etc.).

Differentiated Instruction and Universal Design for Learning

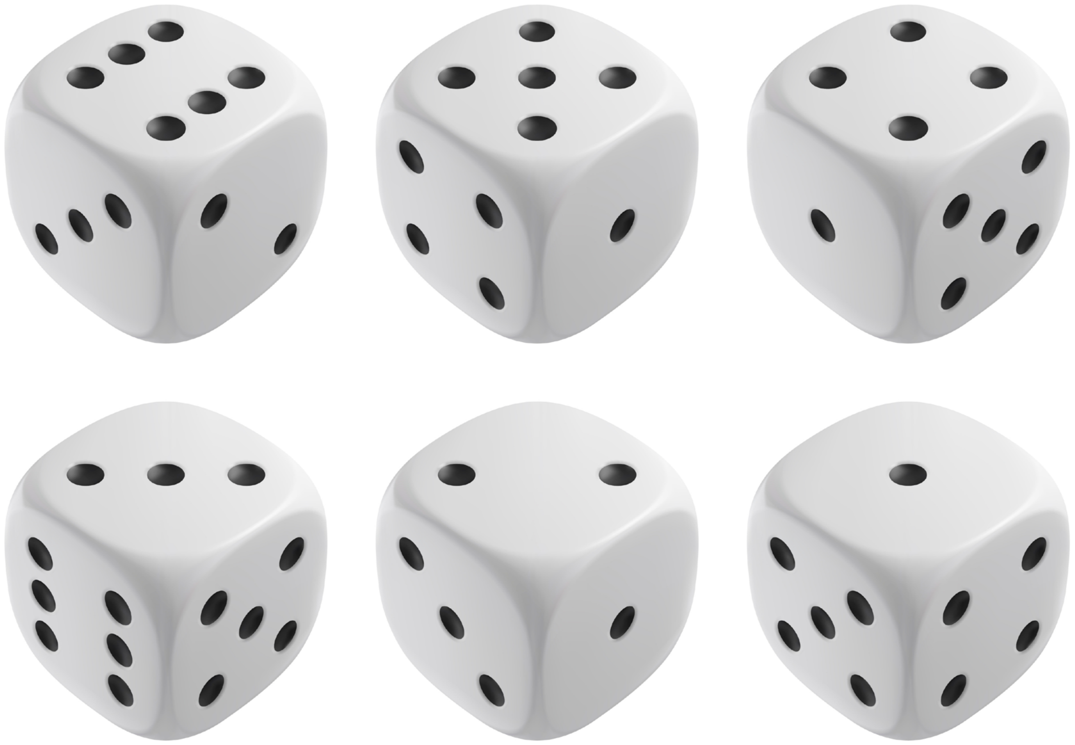
- Give the students the unplugged blocks so they can put them back and make a code out of them.
- Give students the variables before starting the code.
- Propose a beginning of code so that the student can finish it.
- Create some blocks for the student, such as the "multiple of 3" block and the "even number" block.



This block of code steps through the list "List of throws". For each element of the list that is even (2, 4 or 6), the variable "Number of evens" is incremented by 1.

```
define Even numbers
repeat item Length of List of throws time
    if item Throw number of List of throws = 2 or item Throw number of List of throws = 4 or item Throw number of List of throws = 6 then
        change Number of evens by 1
    change Throw number by 1
```

Appendix – Dice



Learning Situation – Grade 6

Title: Little Fish, Little Fish

Duration: 120 minutes

Overview

Students code a video game that involves giving directions to a *sprite* to perform translations on a Cartesian plane in order to "catch another *sprite*".

Overall and Spectific Expectations

Number

B1 Demonstrate an understanding of numbers and make connections to the way numbers are used in everyday life.

- **B1.1** Read and represent whole numbers up to and including one million, using appropriate tools and strategies, and describe various ways they are used in everyday life.

Algebra

C3 Solve problems and create computational representations of mathematical situations using coding concepts and skills.

- **C3.1** Solve problems and create computational representations of mathematical situations by writing and executing efficient code, including code that involves conditional statements and other control structures.
- **C3.2** Read and alter existing code, including code that involves conditional statements and other control structures, and describe how changes to the code affect the outcomes and the efficiency of the code.

Spatial Sense

E1 Describe and represent shape, location, and movement by applying geometric properties and spatial relationships in order to navigate the world around them.

- **E1.3** Plot and read coordinates in all four quadrants of a Cartesian plane, and describe the translations that move a point from one coordinate to another.

Preferred High-Impact Instructional Practices in Mathematics

Problem-Solving Tasks and Experiences

By using a context that is authentic to students, such as creating a video game, students demonstrate motivation and engagement in the activity. Problem solving often requires a lot of trial and error, while making connections with the student's personal experience. Students can then be invited to compare the tasks and determine what is similar and different between them.

Math Conversations

By having math conversations about a concept that is both familiar (video games) and new (coding) to students, teachers are able to see what the student actually understands. Conversations deepen student thinking and challenge students to go beyond their basic knowledge and understanding. Connections and transfers to other strands are noticed during math conversations.

Prior Knowledge and Skills

- Basic knowledge of block-based programming software;
- How to use coordinates properly;
- *Sprite* movement using positive (right or up) and negative (left or down) movements;
- Creating or altering the appearance of a *sprite*;
- Condition blocks;
- The change in size of *sprites*;
- The change of canvas;
- Continuously moving an object using the "repeat infinite" loop;
- The coordinates of a Cartesian plane (four quadrants);
- The number line;
- Negative integers and their effect on an operation or move.

Learning Goals

At the end of this learning situation, the student will be able to:

- create and organize code using knowledge of code conventions and efficiency with the goal of making it into a video game;
- read and represent intergers using the number line;
- plot and read coordinates in the four quadrants of a Cartesian plane in order to correctly position the characters in their game.

Criteria According to the Achievement Chart

Knowledge and Understanding

- The student knows the blocks of block-based programming software and understands how they work.
- The student understands the effect of the negative sign on the location of a number on a number line.
- The student uses coordinates on a Cartesian plane to determine the location of a *sprite* on the canvas.

Thinking

- The student finds solutions to code problems (debugging).
- The student analyzes or plans a coding project using pseudocode.
- The student organizes their code while respecting conventions and taking efficiency into account (use a minimum of blocks in order to reach the desired outcome).

Communication

- The student respects the syntax of the chosen coding software (puts the blocks in the right order and uses control structures that create efficient code).

Application

- The student uses the properties of negative integers to program the movement of a *sprite*.
- The student uses their knowledge of coding to create a game that includes the use of a Cartesian plane and integers.

Materials

- computer with access to the chosen software
- appendix Cartesian Plane

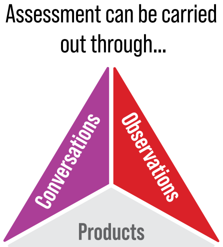
Note: The examples in this learning situation were designed with Scratch.

Mathematical Vocabulary

coding, code, sequential events, concurrent events, repeating events, loop, condition (*if*, *then*, *else*), nested events, Cartesian plane, integer, coordinate, efficiency (in a coding context)

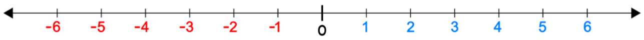
CONTENTS

Before Learning (Warm-Up) (30 minutes)

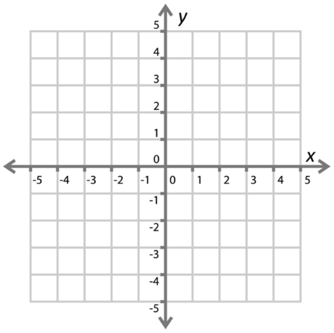


Unplugged Activity to Prepare for Coding

Ask students to stand and imagine themselves at the coordinates (0, 0) of a Cartesian plane. If this is the first time students are introduced to the four quadrants of a Cartesian plane, it would be beneficial to begin with a mini-lesson that shows students that the axes of the Cartesian plane are simply two number lines that intersect at 0, hence the coordinates (0, 0).



If the coordinates of the Cartesian plane have not yet been taught, now is a good time to introduce them, as students will benefit from this new learning immediately. As support, display, if necessary, a Cartesian plane with numbers on the *x* and *y* axes, which will serve as a memory aid throughout the activity.



Ask students to determine, in the plane, where the coordinates (2, 3) is in relation to their current position (0, 0). Ask them for directions to get to (2, 3).

Examples of answers:

- You can take two steps to the right and three steps up.
- You can take three steps up and two steps to the right.
- You can indicate the coordinates and slide diagonally from (0, 0) to (2, 3).

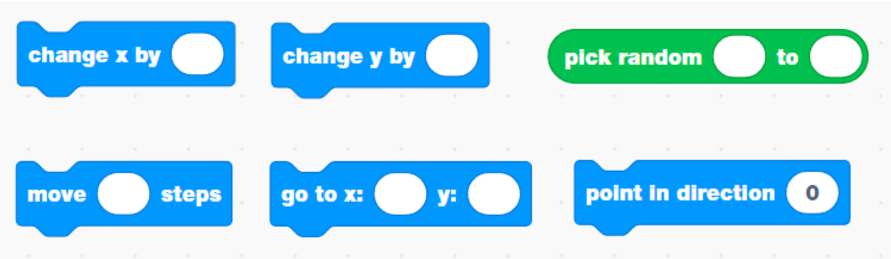
Repeat the exercise, but this time with the coordinates (-2, -3) relative to their initial position (0, 0).

Examples of answers:

- You can take two steps to the left and three steps down.
- You can take three steps down and two steps to the left.
- We can indicate the coordinates and slide diagonally from (0, 0) to (-2, -3)

Point out to students that in everyday life, words such as *to the right* and *up* can be used, but it is also possible to describe movements using positive and negative numbers.

Present students with code blocks similar to these.



Note: For a group of students who are very comfortable with coding and the software in question, the blocks could be presented at the same time. If this is a first exploration of coding or software, the sequence below is suggested.

- Move forward [ ] steps AND turn to [ ].
- Add [ ] to *x* AND add [ ] to y AND go to *x*:[ ] *y*:[ ]
- Random number between [ ] and [ ].

Ask students to share their observations about the blocks. Answers may vary, but the important point to make is that the terminology used in coding is additive (advancing, adding, etc.). Ask students how you can move a *sprite* across the whole Cartesian plane if you can only 'move forward' or 'add'.

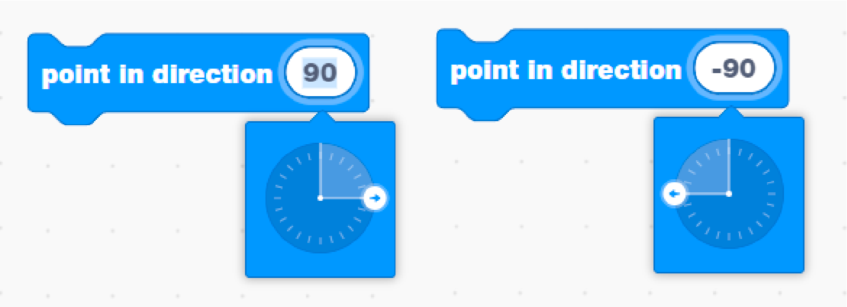
Possible answers

- Always start the *sprite* in the lower left corner in order to "move forward".

This response is interesting because the student realizes that moving up and to the right requires addition. Students' prior knowledge of inverse operations can be used to help them discover that moving left and down requires subtraction.

- You should always use a "point to" block in order to know the direction of movement.

This option is functional, but is much less efficient and risks creating errors in the code. That being said, a student who has difficulty with negative integers could use this strategy to better understand how the number line works. The operation of the "point to" block, by its very nature, introduces students to the effect of the negative sign by using 90 degrees to point right and -90 degrees to point left.



point in direction 90

point in direction -90

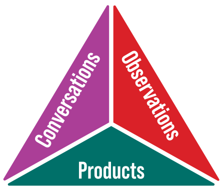
Note: You have to click on 90 in order to enter a direction on the wheel.

- Use positive movements for "up" and "right", and negative movements for "down" and "left".

This answer demonstrates an excellent understanding of the negative sign effect and is the most efficient way to program a move in a Cartesian plane. In coding, you have to use negative numbers to represent downward and leftward movements.

Active Learning (Exploration) (60 minutes)

Assessment can be carried out through...

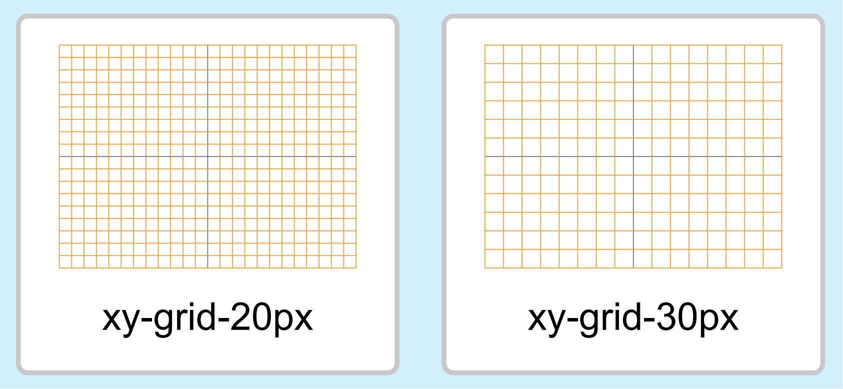


Guided Exploration – Positions on the Cartesian Plane

In order to aim the learning goals, the chosen coding software must have certain features. The software must:

- allow a change of canvas;
- operate with coordinates in the four quadrants of the Cartesian plane;
- allow the student to perform translations by isolating the horizontal (*x*) and vertical (*y*) component.

When students are at their computer with a new blank project on the screen, ask them to choose a canvas that represents a Cartesian plane. It is possible that some Cartesian planes use different scales.

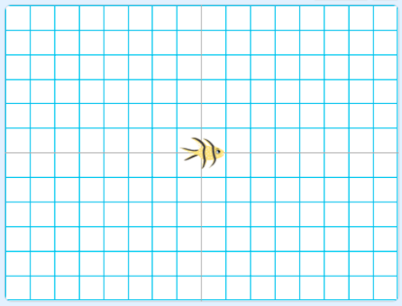


Here are two canvases that have two different scales, either graduations every 20 pixels or every 30 pixels.
We also see that the *x*-axis and the *y*-axis are highlighted with a different color.

Once the canvas is chosen, ask students questions related to scale to ensure understanding.

- What is the typical unit of measurement for graphical elements on a canvas? (pixels)
- How many "steps" (pixels) are needed for the *sprite* to move one square? (The answer depends on the scale. In the example above, the answer could be 20 steps (pixels) or 30 steps (pixels). This answer can be discovered by experimentation.)

Ask the students to choose a *sprite* and place it at the coordinates (0, 0) of the Cartesian plane. In order for it to be at coordinates (0, 0) at the start of the game, the "go to" block could be added at the start of the code (after the green flag).



A *sprite* is placed at coordinates (0, 0) on a Cartesian plane in which the tick marks are 30 pixels apart.

Provide the student with a list of coordinates that their *sprite* needs to go to using only the "add to *x* " and "add to *y* " blocks. Coordinates should be in all four quadrants of the plan; for example, starting from (0, 0), the *sprite* must go successively to the following coordinates:

(-3, 3)

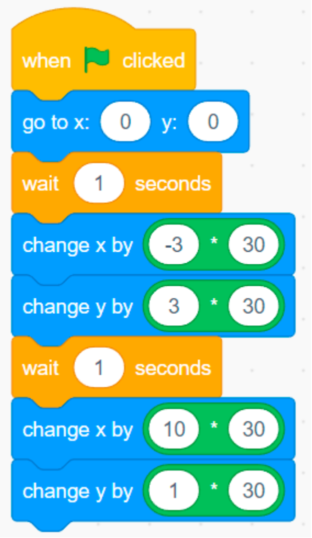
(7, 4)

(-7, -4)

(2, -1)

(0, -3)

The student's task is to determine the translation needed for the *sprite* to travel to the new coordinates. The nature of the task causes the position of the *sprite* to change, so the starting point of the *sprite* is constantly changing. Here is an example of code that respects the first two translations in the list.



```
when ⚑ clicked
go to x: 0 y: 0
wait 1 seconds
change x by -3 * 30
change y by 3 * 30
wait 1 seconds
change x by 10 * 30
change y by 1 * 30
```

We see two translations in this code. The coordinates of the first starting point are (0, 0). The first translation is the same as the coordinates of the final position, which is (-3, 3).

For the second translation, the new starting point (-3, 3) must be considered. So the translation in *x* is 10 and the translation in *y* is 1.

This last movement is compared to the integers located on a number line; for example, to go from −3 to +7, a movement of 10 units to the right is necessary (horizontal number line). The same principle applies for a movement in *y*, which is upwards (vertical number line). Encourage the student to use a number line in order to better define the desired movement.

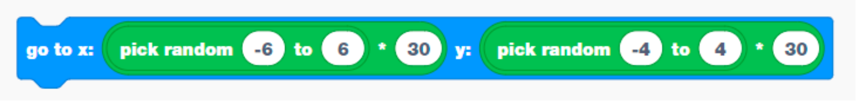
Note that the operation blocks that multiply each number by 30 are essential in order to respect the scale of the Cartesian plane which, in this example, includes a scale of 30 pixels.

Help students code the movements of the *sprite* according to various coordinates of a Cartesian plane.

Following the guided exploration, ask students if it is efficient (and realistic) to code all possible positions of the *sprite* on the canvas in the context of a game. The answer is no. Ask them for changes that might make the code more efficient or more functional.

Possible answers

- We could use the "go to (*x*) (*y*)" block, which does the translation for us without us having to calculate it.
- One could use the "go to random position" block, which eliminates the predictability of the *sprite*'s location.
- The 30 pixel scale is useful for the Cartesian plane canvas, but with a different canvas one can use the current coordinates instead of always multiplying by 30, which would allow for more accurate positions.



```
go to x: (pick random (-6) to (6) * (30)) y: (pick random (-4) to (4) * (30))
```

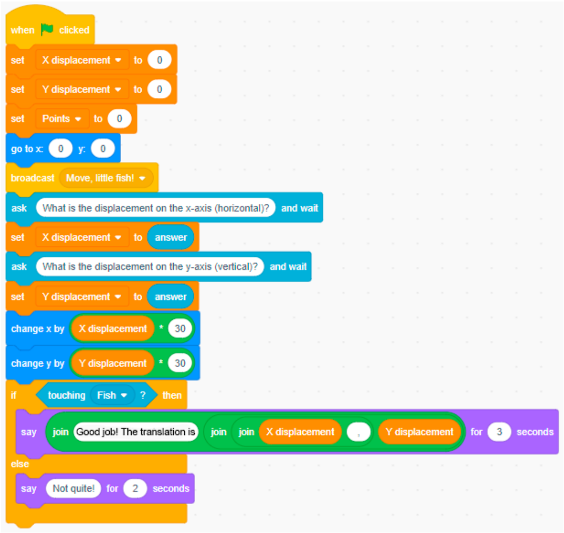
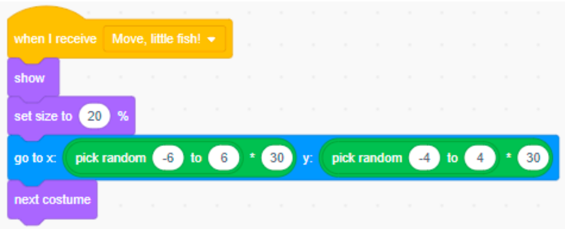
This block would allow a *sprite* to move at random coordinates on a Cartesiane plane with a scale of 30 pixels.

In the example game below, some answers can be found in the code.

Programming an Interactive Game

Challenge students to create a game to be shown to a peer. The objective of the game is to test the peer's knowledge of the description of translations in the four quadrants of the Cartesian plane. The game will consist of moving a *sprite* using translation vectors (coordinates) so that it touches various objects.

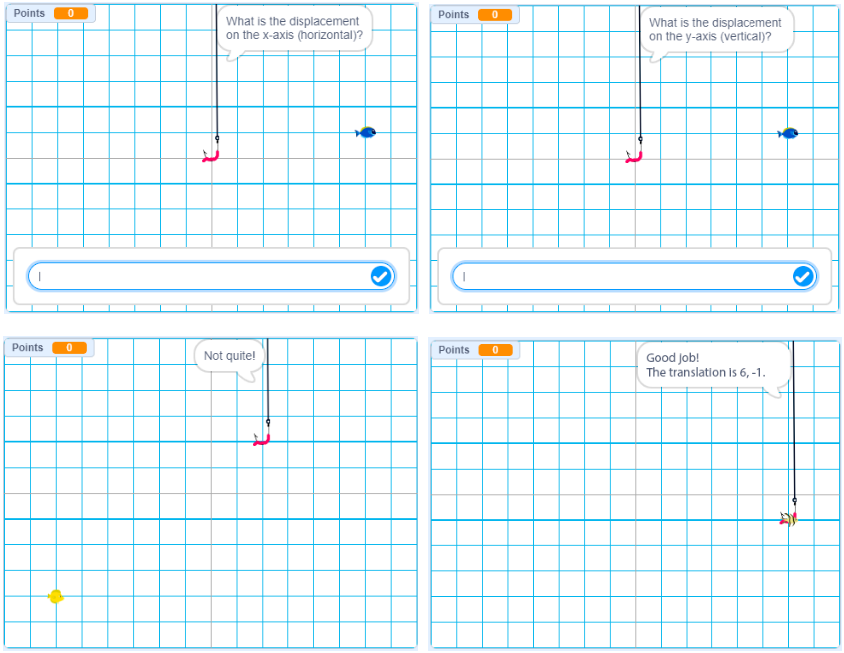
There are several possibilities of games that could test the understanding of these concepts. Here is an example of a fishing game that the student could produce:

HOOK *SPRITE* CODE	FISH *SPRITE* CODE
 when 🏁 clicked set X displacement ▾ to (0) set Y displacement ▾ to (0) set Points ▾ to (0) go to x: (0) y: (0) broadcast Move, little fish! ▾ ask (What is the displacement on the x-axis (horizontal)?) and wait set X displacement ▾ to (answer) ask (What is the displacement on the y-axis (vertical)?) and wait set Y displacement ▾ to (answer) change x by ((X displacement) * (30)) change y by ((Y displacement) * (30)) if <touching Fish ▾ ?> then say (join (Good job! The translation is) (join (join (X displacement) (,)) (Y displacement))) for (3) seconds else say (Not quite!) for (2) seconds	 when I receive Move, little fish! ▾ show set size to (20) % go to x: (pick random (-6) to (6) * (30)) y: (pick random (-4) to (4) * (30)) next costume

Note: In this code, we see the use of variables that determine the movements necessary for the hook to "catch" the fish. We also see the use of a "message". Messages allow *sprites* to interact more and can be very useful in the context of coding a game. In this case, when the message "Move, little fish!" is sent, the fish *sprite* receives the message and executes the code.

We also see the use of the "group" block, which makes the code less cumbersome by avoiding using four different "say" blocks to communicate with the user.

This is what the game interface might look like:



What is the displacement on the x-axis (horizontal)?

What is the displacement on the y-axis (vertical)?

Not quite!

Good job!
The translation is 6, -1.

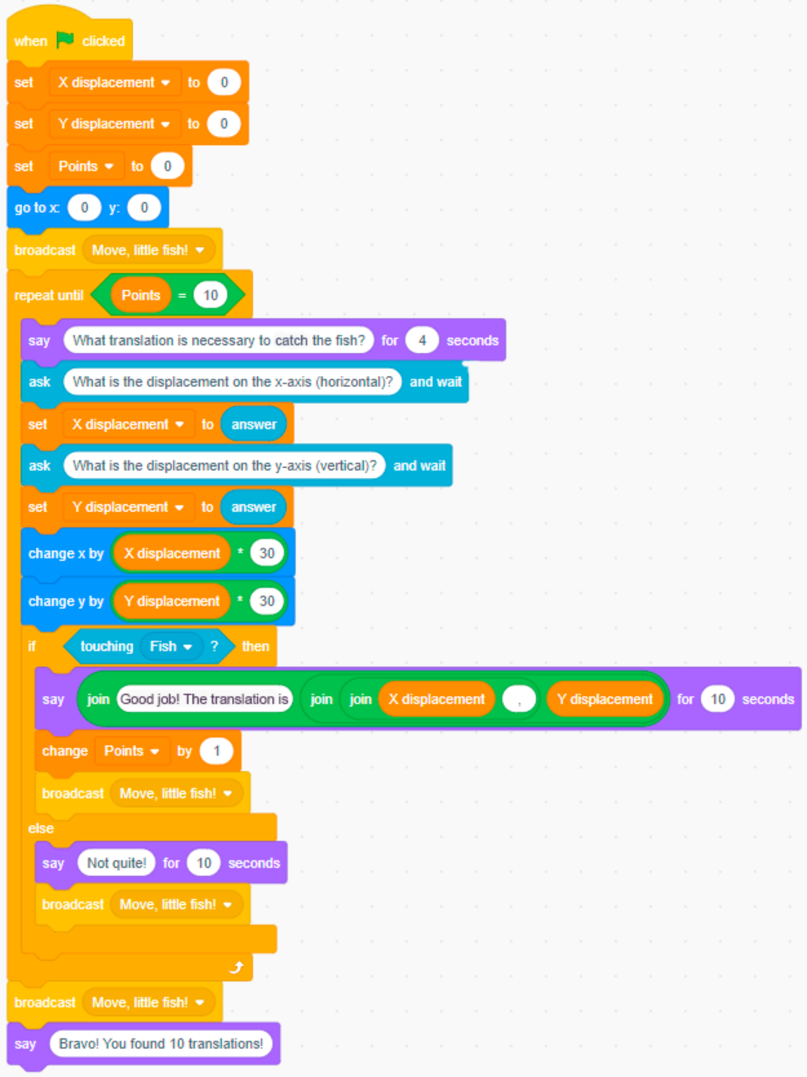
Note: Screenshots were taken of the full-screen program interface at various moments in the game.

Go Further – The Fishing Game

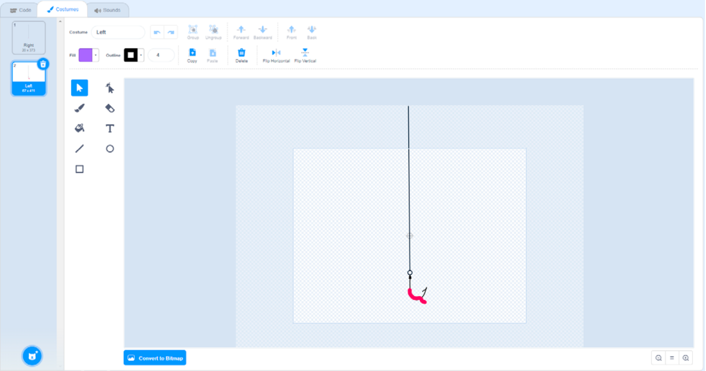
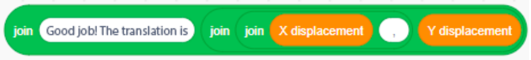
There are several ways to make the game more interactive and fun. Encourage students to make the game as interactive as possible. Here are some suggestions:

- Add a variable which makes it possible to count the score and make the code repeat until a certain number of points is reached.
- Use a Cartesian plane with a different scale to create an easier or more difficult level.
- Add a time element, like a stopwatch that would count down.
- Code the fish to remain in place for a specific number of seconds.

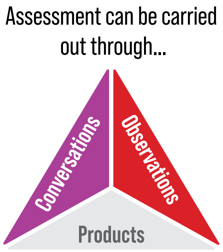
Here is an example of code that incorporates the scoring and repeating elements:



```
when ⚑ clicked
set X displacement ▾ to 0
set Y displacement ▾ to 0
set Points ▾ to 0
go to x: 0 y: 0
broadcast Move, little fish! ▾
repeat until Points = 10
    say What translation is necessary to catch the fish? for 4 seconds
    ask What is the displacement on the x-axis (horizontal)? and wait
    set X displacement ▾ to answer
    ask What is the displacement on the y-axis (vertical)? and wait
    set Y displacement ▾ to answer
    change x by X displacement * 30
    change y by Y displacement * 30
    if touching Fish ▾ ? then
        say join Good job! The translation is join join X displacement , Y displacement for 10 seconds
        change Points ▾ by 1
        broadcast Move, little fish! ▾
    else
        say Not quite! for 10 seconds
        broadcast Move, little fish! ▾
    ⤴
broadcast Move, little fish! ▾
say Bravo! You found 10 translations!
```

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student cannot move their *sprite* to the given coordinates.	Remind the student that a pixel is very small and that a scale has to be used to obtain the desired translation. Check that the student has coded the starting position as (0, 0), and that the *sprite* is placed back in that location to resume the code.
The student does not know how to alter or create a *sprite*.	Review with the student the way the software works and review how to add a *sprite*. Ask the student if something on the page might help.  Code Costumes Sounds Costume Left Right Group Ungroup Forward Backward Front Back Fill Outline Copy Paste Delete Flip Horizontal Flip Vertical Convert to Bitmap The hook used in the example is drawn by hand. Students can choose existing *sprites* or create or modify a *sprite* in the "costumes" menu.
The student is unable to use variables well.	Question the student to check their understanding of the "variable" block. • Could you describe to me what a variable is? • Are there any values that change in your code? It can also be beneficial to make all the variables visible on the interface so that the student can take into account the value associated with each of them.
The student does not use the "group" block to create complex sentences with variables.	The "group" block can be a bit intimidating at first. Pseudocode can help support the understanding of its purpose. In the code example above, the "group" block is used to communicate the translation to the student. The pseudocode might look like this: Say, "Good job! The translation will be (variable move in *x*), (variable move in *y*)." A "join" block with four elements (ovals) is then created. Example:  join (Good job! The translation is) (join (join (X displacement) (,)) (Y displacement))
The student has difficulty using conditional structures.	Have the student verbalize the intent of their code or write pseudocode. Find the places where you see words indicating conditions (*if*, *then*, *until*, *when*).

Review (30 minutes)



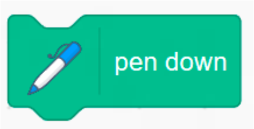
Have a learning fair with the students to try out the games. Ask them to note the similarities and differences between the games they tried and their own. Encourage peer feedback during playtesting.

Guide reflection through questioning.

- What are the main differences between the codes? Is there more than one way to code the desired outcome? (Yes, there are many ways to write code. It's likely that the codes in the class will be similar, but not identical.)
- Would it be possible to make the game easier (for example, by adding numbers to the axes) or more difficult (for example, by removing the Cartesiane plane and requiring position prediction)?
- Have you noticed elements, in other codes, that you would like to integrate into yours (for example, more precise visual or textual elements, more efficient codes, specialized blocks, such as booleans or conditional loops)?

Consolidation of Learning

Have students program a trivia game in which a *sprite* must travel to specific coordinates. The student would first have to enter the necessary translations in order to move from one coordinate to another. Adding a block to trace the path could reveal a shape or mystery message indicating that the student has successfully moved their *sprite* to the correct places. During the game trial, encourage peer feedback.



The "pen down" block is an example of a block that allows you to trace the *sprite*'s path.
This block is part of the Scratch coding software. In other software, this block type might have a different name.

CONSIDERATIONS

Links to Other Curriculum Expectations

Number

B2.1 Use the properties of operations, and the relationships between operations, to solve problems involving whole numbers, decimal numbers, fractions, ratios, rates, and whole number percents, including those requiring multiple steps or multiple operations.

A multi-step problem can quickly become a multi-line code. Efficiency can therefore come as a support in order to simplify a more complex problem, for example, by using loops, variables or custom blocks.

Spatial Sense

E1.3 Plot and read coordinates in all four quadrants of a Cartesian plane, and describe the translations that move a point from one coordinate to another.

The Cartesian plane is often used to show position and movements in visual coding contexts. It is therefore possible to ask the student to make several movements using the coordinates of the Cartesian plane. The student can later make their code more efficient by organizing the moves and using custom loops and blocks.

Financial Literacy

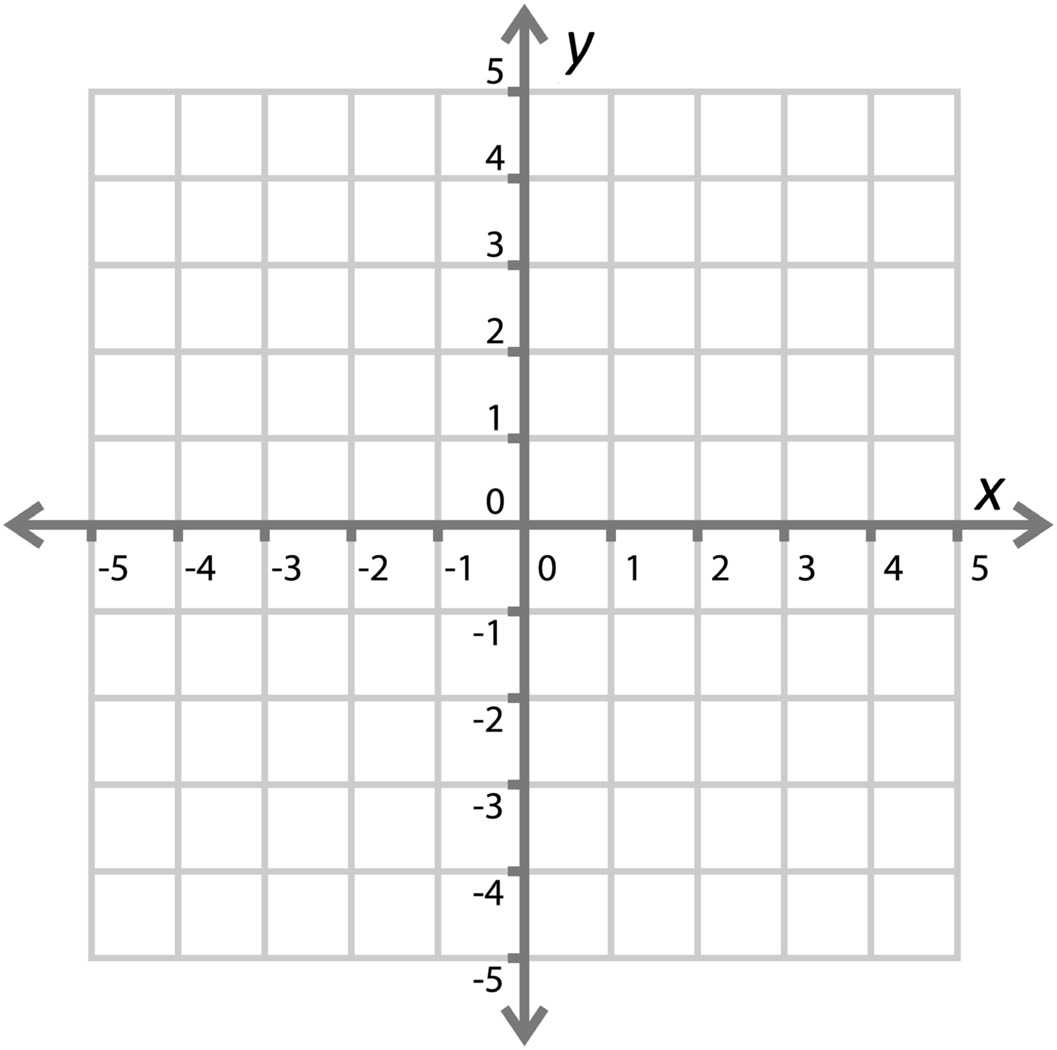
F1.4 Explain the concept of interest rates, and identify types of interest rates and fees associated with different accounts and loans offered by various banks and other financial institutions.

Interest scenarios, especially compound interest, can easily become multiple lines of code. Using variables, custom blocks, and generalizations can simplify code (and make a complex financial situation easier to understand).

Differentiated Instruction and Universal Design for Learning

- Have the student create some pseudocode before they even begin to create a blueprint for their programming.
- Determine the variables with the student from the beginning.
- Provide incomplete code for the student to alter.
- Provide cheat sheets with the functions of the different blocks.
- Invite students to work in teams, either to encourage peer learning or to foster direct and personalized instruction according to their needs.
- Encourage students to create levels in their game so that the difficulty gradually increases (for example, fish are not stationary, there are objects to avoid).

Appendix – Cartesian Plane



Learning Situation – Grade 7

Title: Shoe Sizes
Duration: 180 minutes

Overview

In this learning situation, the student collects the shoe sizes of the students in their class in order to suggest an inventory of shoes for preteens at the community shoe store. From the data, the student creates code using block-based programming software in order to find the mean and the median of the shoe sizes of the students in the class. The student observes that adding or subtracting data can affect measures of central tendency.

Overall and Spectific Expectations

Data

D1. Manage, analyse, and use data to make convincing arguments and informed decisions, in various contexts drawn from real life.

D1.5 Determine the impact of adding or removing data from a data set on a measure of central tendency, and describe how these changes alter the shape and distribution of the data.

Algebra

C3. Solve problems and create computational representations of mathematical situations using coding concepts and skills.

C3.1 Solve problems and create computational representations of mathematical situations by writing and executing efficient code, including code that involves events influenced by a defined count and/or subprogram and other control structures.

C3.2 Read and alter existing code, including code that involves events influenced by a defined count and/or subprogram and other control structures, and describe how changes to the code affect the outcomes and the efficiency of the code.

Preferred High-Impact Instructional Practices in Mathematics

Direct Instruction

During Warm-Up, the teacher questions the students about the mode, the median and the mean in order to activate their prior knowledge. In light of student responses, vocabulary under study is introduced such as *measures of central tendency* and *subprogram*. The teacher shows a subprogram (pseudocode for mode) and asks students about the pseudocode. Students voice their opinions/reflections in a think-pair-share and then work in guided practice to create pseudocodes for the median and mean.

Small-Group Instruction

During Exploration, small-group instruction is done, about 10 minutes at a time, with predetermined groups to converse and guide the students in creating their code. This is a great time to talk about conditional statements and how these blocks affect a code.

Flexible Groupings

Communication and collaboration are assets in tasks like this. By working in groups, students move around and find a place to work with their peers. The teacher circulates among the students and supports them as needed. The teacher questions students and pushes their thinking as they create to collect data and to determine the mean and the median.

Prior Knowledge and Skills

To be able to carry out this learning situation, the student must:

- understand what pseudocode is;
- know what measures of central tendency are;
- be able to use block-based programming software.

Learning Goals

At the end of this learning situation, the student will be able to:

- determine measures of central tendency using block-based programming software;
- repeatedly use blocks of conditional statements as well as operator blocks;
- write efficient pseudocodes;
- recognize the impact of adding or removing certain data on measures of central tendency.

Criteria According to the Achievement Chart

Thinking

- The student writes pseudocodes accurately to collect data, determine the mean, and find the median of the collected data.
- The student accurately corrects syntax and logic errors in their codes (debugging).
- The student explains the impact of adding or removing data on the mean and the median.

Communication

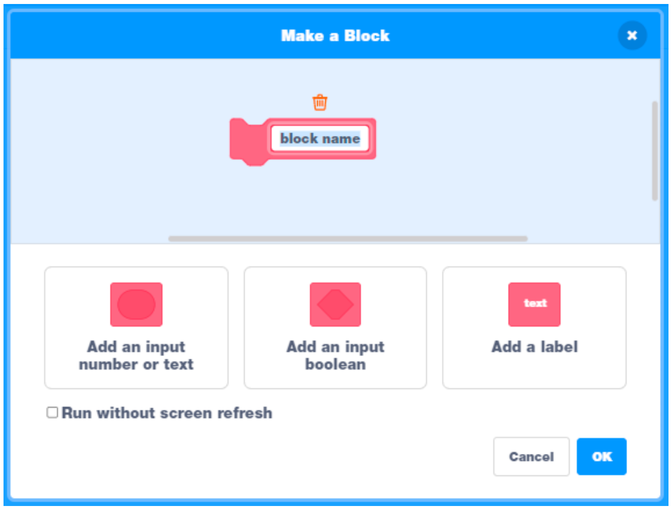
- The student coherently explains the effect of certain blocks, such as operator blocks and control blocks.
- The student clearly explains the code of their peers.

Knowledge and Understanding

- The student understands the category of “my blocks” to create custom blocks* and then uses them in their code.
- The student uses variables in their code.

Application

- The student creates an effective code to collect the shoe sizes of the students in the class.
- The student consistently adds blocks of conditional statements into their codes.



* **Custom Blocks:** This screenshot shows the menu for creating a new block in the Scratch coding software. By creating a new block, one can make the code more efficient by summarizing a sequence of code to be repeated at various places in the code into a single block. Creating blocks in other block-based coding software might have an interface that differs from this one.

Materials

- block-based programming software
- paper
- pencil
- appendix 1 (Warm-Up)
- appendix 2 (Pseudocode)

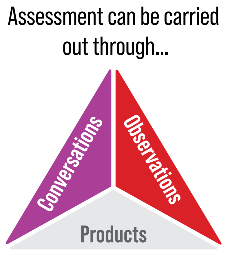
Note: The examples in this learning situation were designed with Scratch.

Mathematical Vocabulary

measures of central tendency, mode, median, mean, subprogram, pseudocode

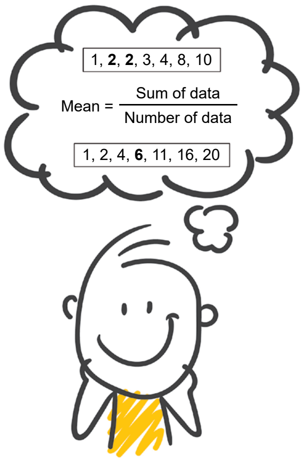
CONTENTS

Before Learning (Warm-Up) (15 minutes)



Ask the students to observe, as a class, the picture below (appendix 1 (Warm-Up)) while trying to imagine the mathematical problem that could be associated with it.

Show students the picture below. Inform them that the two pictures are related. Ask them again to ask questions and try to determine the problem associated with the picture. Guide questioning so that students ask questions about measures of central tendency (mean, median, and mode).

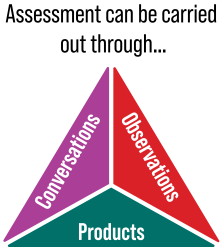


Example questions:

- Why do you think some numbers are in bold?
- What math could be related to shoes?
- Are all the shoes in the picture the same or different? How does this affect your interpretation of the problem?

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student does not discuss measures of central tendency.	What does the formula represent? Have you ever used a formula like this? In what context? Why do you think the number 2 is written in bold?
The student has no idea what the shoes might represent.	How many shoes do you see? What do you think are the shoe sizes? How can shoes differ from each other?
The student does not know the measures of central tendency.	Review with students the concepts of mode, median and mean.

Active Learning (Exploration) (150 minutes)



Getting Ready for Coding

Explain to the students that data concerning the shoe sizes of the students in the class are to be collected. Using this data, students will create codes to help find the mean and median of shoe sizes in block-based programming software.

For thought

- What data are we looking for?

  We are looking for the shoe sizes of the students in the class.

- How can data collection be automated?

  Using a code (in blocks or in a spreadsheet), visiting a website where you can create surveys, filling out a form, etc.

- Would it be possible to create a code for data collection? What will the code need to do?

  The code should ask for a person's shoe size, add that shoe size to a list, then ask someone else until all the data is collected (a pseudocode could help with this step).

- Is it easier to code for mean, median or mode?

  Some students might say that the median or the mode are easier to code because it is simply a matter of finding a number in a list, while the mean requires a mathematical operation. In reality, the code to sort the list of data in ascending order is much more demanding than the code to do mathematical operations. It would be interesting and relevant to ask this question again at the time of the Review.

- What does the code need to do to calculate or recognize each of the measures of central tendency?
  - For the mean, the code has to calculate the sum of all the data, then divide this sum by the number of data in the list.
  - For the median, the code has to sort the data in ascending order and recognize the central value, either by finding the value in the center of the list (odd number of data) or by calculating the mean of the two values in the center of the list (even number of data).
  - For the mode, the code has to sort the data in ascending order, then compare the values to determine which one is found most often in the list.

Present the pseudocode below to the class to help students understand what a pseudocode is.

```
Variables: nb1
           nb2
           sum
Begin
    Set nb1 = 0.
    Set nb2 = 0.
    Set sum = 0.
    Define the value of nb1.
        Ask the question "What is the value of nb1?".
        The answer becomes the value of nb1.
    Define the value of nb2.
        Ask the question "What is the value of nb2?".
        The answer becomes the value of nb2.
    Sum = nb1 + nb2
End
```

- What are the variables? What do they represent?

  The variables are:

  - nb1: the value of the first piece of data collected;
  - nb2: the value of the second piece of data collected;
  - sum: the sum of all the data.

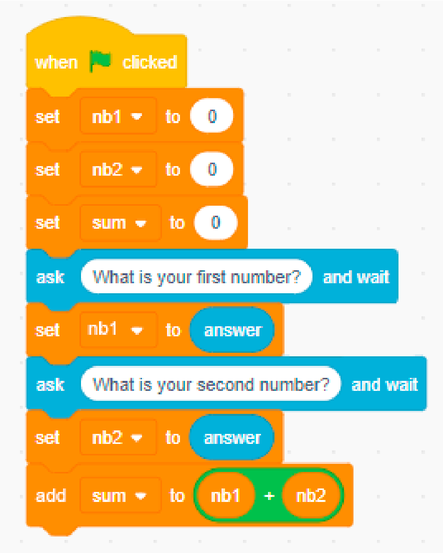
- What is the role of the three "set to 0" blocks at the beginning of the code?

  Without the "set to 0" blocks, any variables that were set the last time the code was executed remain active. This could have undesirable outcomes from the code. It's a bit like pressing the "CE" key on a calculator.

- What would be the code, in block-based programming software, for this pseudocode?

  Invite students to create the code that the pseudocode represents. Codes may vary. There are many ways to represent this pseudocode using block-based coding.

Sample code:



```
when ⚑ clicked
set nb1 ▾ to 0
set nb2 ▾ to 0
set sum ▾ to 0
ask What is your first number? and wait
set nb1 ▾ to answer
ask What is your second number? and wait
set nb2 ▾ to answer
add sum ▾ to nb1 + nb2
```

Note: In this part of the learning situation, the students make the blocks their own by means of which the questions are asked and the value of the variables is modified. If the students have already used the coding software in question, it would be possible to go directly to the next part of the activity, which is the data collection. This activity can also be used as a diagnostic assessment to determine if students have what they need to move on to data collection.

Take the time to circulate among the students and listen to their conversations. Based on their observations and your active listening, question them, if necessary, to guide them or better understand their reasoning.

Coding to Collect Data

Ask the students to create a pseudocode to collect data from all the students in the class. Students can start from the code prepared in the previous step or start a new code.

Pseudocode example:

Variables: * number of data to enter

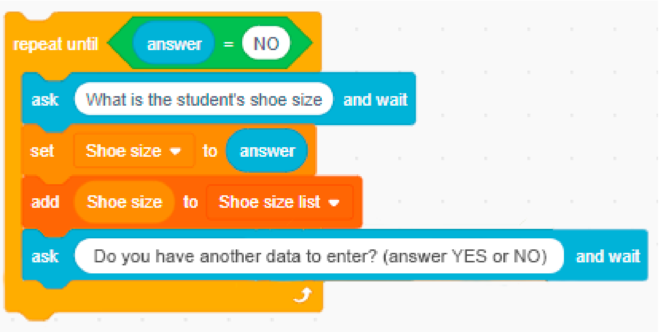
* shoe size

* shoe size data list

```
Begin
     Repeat for each data.
          Ask for shoe size.
          Add the answer to the list of shoe sizes.
          Ask if there is more data to enter.
          If yes – repeat
          If not – end
```

Once the pseudocode is complete, have students create their code to collect data related to students' shoe sizes, then compile it into a list.

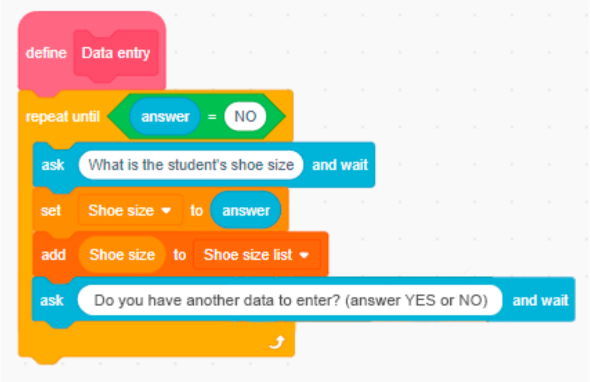
Sample code:



```
repeat until answer = NO
    ask What is the student's shoe size and wait
    set Shoe size to answer
    add Shoe size to Shoe size list
    ask Do you have another data to enter? (answer YES or NO) and wait
```

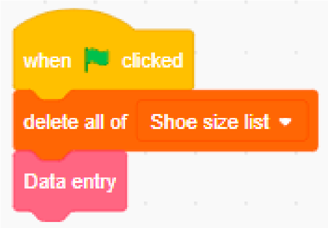
Note: Several block-based coding software make block customization possible. If so, students can create a block (custom block) and define it with their code. This way the main code becomes less cluttered.

Example of a custom block:



```
define Data entry
repeat until answer = NO
    ask What is the student's shoe size and wait
    set Shoe size to answer
    add Shoe size to Shoe size list
    ask Do you have another data to enter? (answer YES or NO) and wait
```

Once this block is created, the main code would look like this:



```
when 🏴 clicked
delete all of Shoe size list
Data entry
```

Coding to Analyze Data

Ask students what the computer has to do to find the average shoe size from the data collected. If necessary, the starting image with the mean formula could be redisplayed in order to activate the students' prior knowledge. The mean formula can also be transformed into pseudocode. Invite them to write instructions for the computer to calculate the mean of the data collected.

Here is an example of pseudocode:

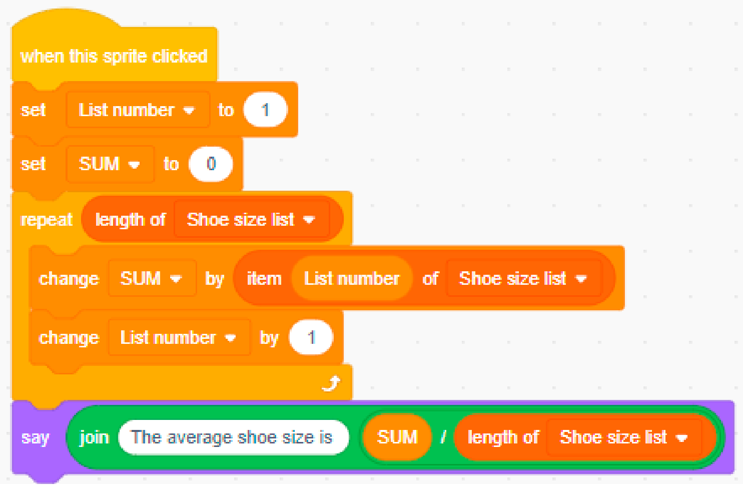
Variables: * list number (or list element)

* sum

* shoe size data list

```
Begin
    Set list number to 1 (we start with the first element of the list).
    Set sum to 0.
    Repeat for list length (number of items).
        Add (element 1) from list to sum.
        Add 1 to list number.
    Say mean = sum/list length.
End
```

Sample code:

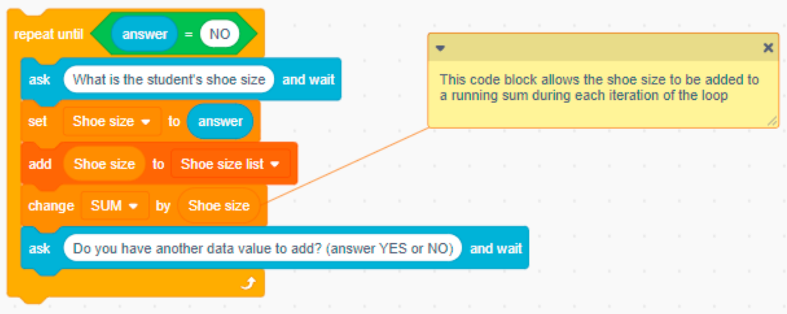


```
when this sprite clicked
set List number to 1
set SUM to 0
repeat length of Shoe size list
    change SUM by item List number of Shoe size list
    change List number by 1
say join The average shoe size is SUM / length of Shoe size list
```

Ask students the following questions:

- What happens if I add shoe sizes to my list?
- Will the mean be the same? Why?
- Could this code be made even more efficient?

Note: This code is structured in this way in order to separate the steps and to encourage student questioning and exploration. A simpler version of this code would see the list items appended to the value of the "sum" variable at data collection time, eliminating that whole step. This code would look like this:



```
repeat until answer = NO
    ask What is the student's shoe size and wait
    set Shoe size to answer
    add Shoe size to Shoe size list
    change SUM by Shoe size
    ask Do you have another data value to add? (answer YES or NO) and wait
```

This code block allows the shoe size to be added to a running sum during each iteration of the loop

Invite the students to create a pseudocode and a code to find the median of the shoe sizes of the students in the class.

Possible questions:

- What are the steps to determine the median?
- Is there a block that can place the data in ascending order? Can we create one? (Students may mention that the elements of the list of shoe sizes must be sorted in ascending order in order to find the median.)

Pseudocode example:

Variables: * central value 1

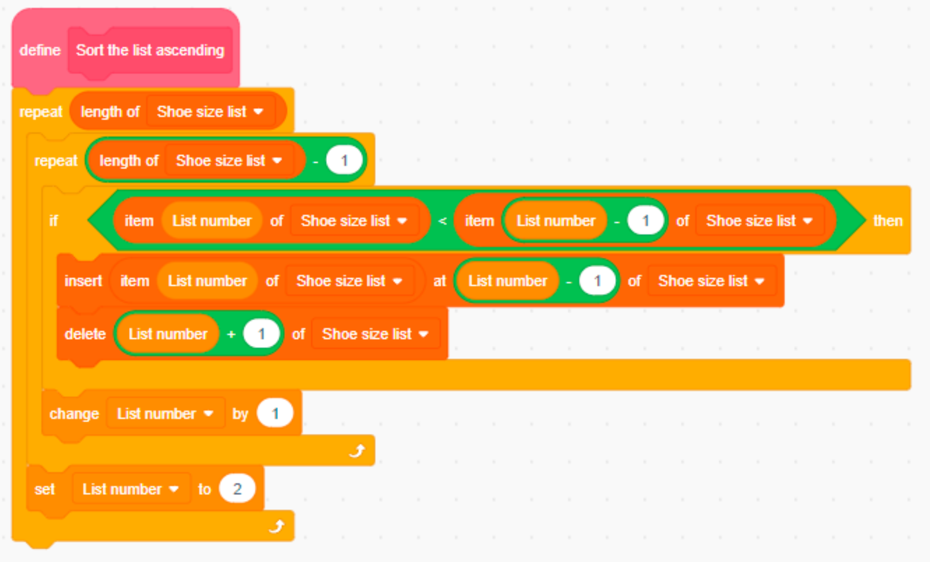
* central value 2

* shoe size data list

```
Begin
Put "list of shoe sizes" in ascending order.
    If the list has an even number of elements
        Identify the element corresponding to (length of the list /2).
            Set the value of the element to ("central value 1").
        Identify the element corresponding to ((length of list /2) + 1).
            Set the value of the element to ("central value 2").
        Set median to (("middle value 1" + "middle value 2")/2).
    ELSE
        Put median to the element corresponding to ((length of the list + 1) /2).
End
```

Note: If the list contains an even number of data, point out to students that calculating the median takes the same form as calculating the mean. Essentially, the mean of the two central values is found.

Here is a sample code to place the data list in ascending order. Again, the code has been turned into a custom block. This step is helpful, but not necessary.



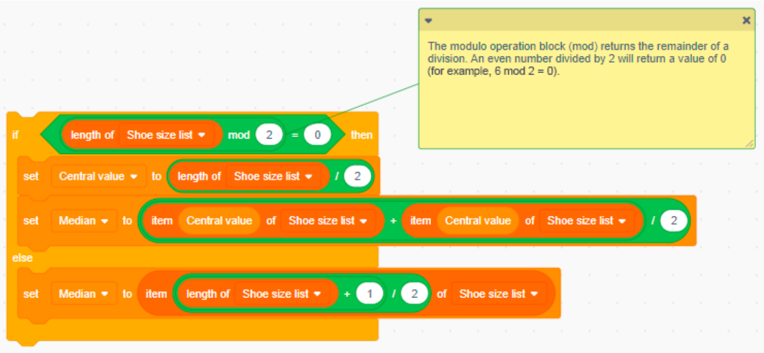
```
define Sort the list ascending
repeat length of Shoe size list
    repeat length of Shoe size list - 1
        if item List number of Shoe size list < item List number - 1 of Shoe size list then
            insert item List number of Shoe size list at List number - 1 of Shoe size list
            delete List number + 1 of Shoe size list
        change List number by 1
    set List number to 2
```

Short explanation of the logic of the code:

This code works by comparing a value in the list with the neighbouring value. If the value is smaller, the positions of the list elements are reversed. The code is repeated the same number of times as there are elements in the list to ensure that each value is smaller than the next value.

Note: Ordering numbers in ascending order is a complex problem for the computer, which must compare the data values and then order them. This kind of problem lends itself very well to an Internet search. There are several communities and a range of existing projects that can answer questions from students and staff. By modelling this kind of resourcefulness, students become more comfortable doing the same, and learn more about what is possible with block-based coding software.

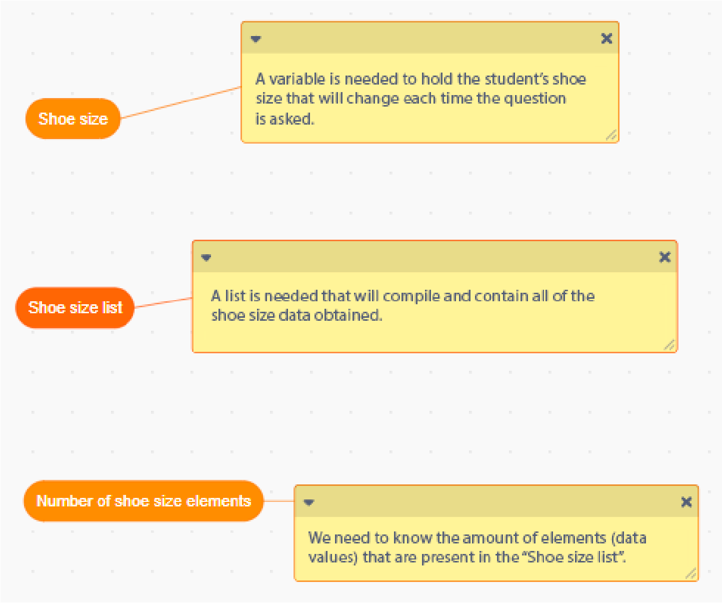
Sample code to find the median:

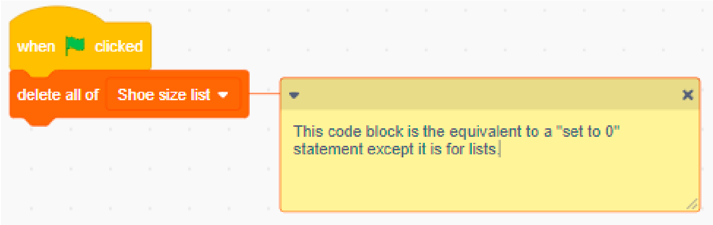
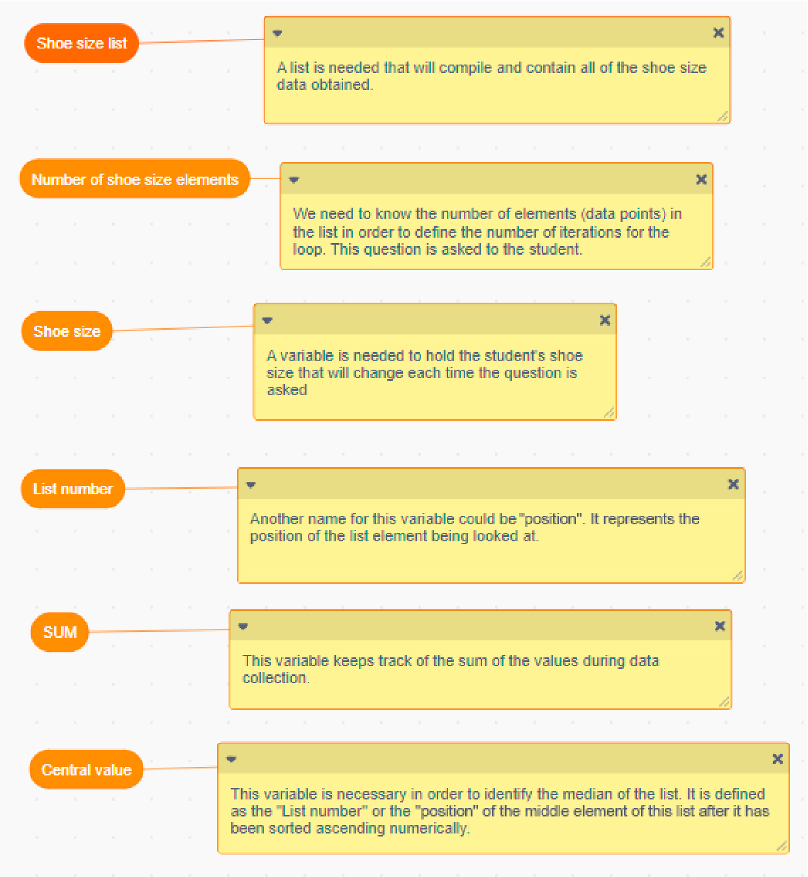


```
if length of Shoe size list mod 2 = 0 then
    set Central value to length of Shoe size list / 2
    set Median to item Central value of Shoe size list + item Central value of Shoe size list / 2
else
    set Median to item length of Shoe size list + 1 / 2 of Shoe size list
```

The modulo operation block (mod) returns the remainder of a division. An even number divided by 2 will return a value of 0 (for example, 6 mod 2 = 0).

Ask students the following questions: What happens if I add shoe sizes to my list? Will the median stay the same? Why?

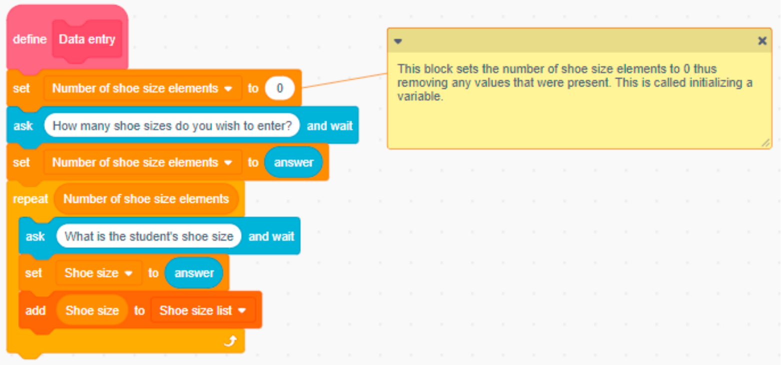
POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student is unable to enter all the sizes of the students' shoes in a list.	In which block can you ask a question? What question can you ask here? What variables do you need?  Shoe size: A variable is needed to hold the student's shoe size that will change each time the question is asked. Shoe size list: A list is needed that will compile and contain all of the shoe size data obtained. Number of shoe size elements: We need to know the amount of elements (data values) that are present in the "Shoe size list".

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student forgets to reset the variables to zero.	What happens when your variables are not reset at the start of code? What is the effect on data entry in the list?  when 🏳 clicked delete all of Shoe size list ▾ This code block is the equivalent to a "set to 0" statement except it is for lists. Ask the student to return to see the pseudocode that was analyzed beforehand. This pseudocode also contains "set to 0" blocks. Encourage them to find the connection between these blocks in the two codes.
The student does not use variables correctly.	Ask the student what the variables mean. Understanding the meaning of the variables used is necessary for understanding the code. The student may not understand the names of the variables, so encourage them to rename them with terms familiar to the context of the problem.  Shoe size list: A list is needed that will compile and contain all of the shoe size data obtained. Number of shoe size elements: We need to know the number of elements (data points) in the list in order to define the number of iterations for the loop. This question is asked to the student. Shoe size: A variable is needed to hold the student's shoe size that will change each time the question is asked List number: Another name for this variable could be "position". It represents the position of the list element being looked at. SUM: This variable keeps track of the sum of the values during data collection. Central value: This variable is necessary in order to identify the median of the list. It is defined as the "List number" or the "position" of the middle element of this list after it has been sorted ascending numerically.

<table>
<tr><th>POSSIBLE OBSERVATIONS</th><th>POSSIBLE INTERVENTIONS</th></tr>
<tr><td>The student fails to create new blocks (custom blocks).</td><td>

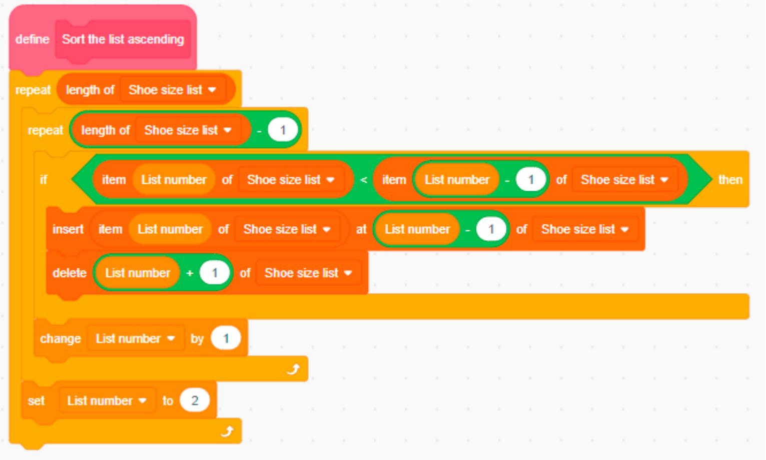
Which block category can help you create new blocks? What is its color?

Ask the student why we create new blocks (to make the codes more efficient by being shorter or to make the code easier to analyze or debug by isolating certain sequences from the main code).



```
define Data entry
set Number of shoe size elements ▾ to 0
ask How many shoe sizes do you wish to enter? and wait
set Number of shoe size elements ▾ to answer
repeat Number of shoe size elements
    ask What is the student's shoe size and wait
    set Shoe size ▾ to answer
    add Shoe size to Shoe size list ▾
```

This block sets the number of shoe size elements to 0 thus removing any values that were present. This is called initializing a variable.



```
define Sort the list ascending
repeat length of Shoe size list ▾
    repeat length of Shoe size list ▾ - 1
        if item List number of Shoe size list ▾ < item List number - 1 of Shoe size list ▾ then
            insert item List number of Shoe size list ▾ at List number - 1 of Shoe size list ▾
            delete List number + 1 of Shoe size list ▾
        change List number ▾ by 1
    set List number ▾ to 2
```

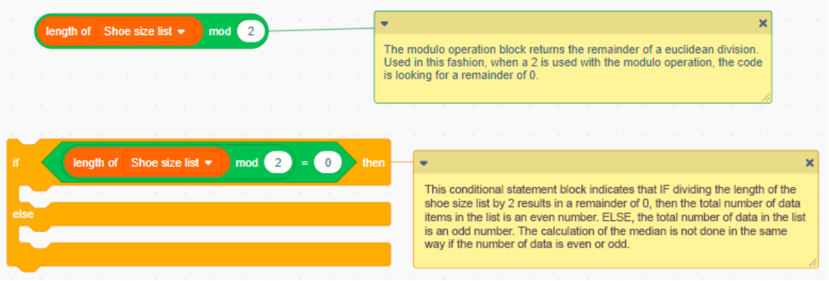
</td></tr>
<tr><td>The student fails to create a code to calculate the mean.</td><td>

Ask the student to review their pseudocode for the mean. Encourage them to locate the calculation of the mean in their pseudocode, then identify the corresponding blocks in their code. If there are blocks missing, that would be the first step.

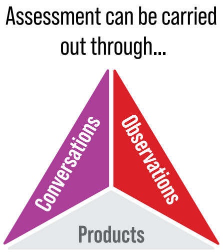
Review with the student the meaning of the blocks and variables. Are the right variables in the right places in the formula?

If necessary, ensure that the values of the variables are displayed on the screen. This way, the student has a visual cue of the values and is more likely to be able to detect if there is an error.

</td></tr>
</table>

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student fails to create a code to order the data in ascending order.	Ordering data in ascending order using blocks can be logical. Ask the student what to do to order a list in ascending order (for example, determining if each value is smaller than the next). How could we make the computer understand it? Show the student the operators "greater than" and "less than" as a clue. Invite the student to search the Internet to see if this problem already has a solution instead of creating all new code.
The student fails to create a code to find the median.	Ask the student the meaning of certain operator blocks. What does the *if, then, else* conditional statement block mean? How do you usually find the median of numbers? In this code, what variables are needed to find the median?  length of Shoe size list mod 2 The modulo operation block returns the remainder of a euclidean division. Used in this fashion, when a 2 is used with the modulo operation, the code is looking for a remainder of 0. if length of Shoe size list mod 2 = 0 then else This conditional statement block indicates that IF dividing the length of the shoe size list by 2 results in a remainder of 0, then the total number of data items in the list is an even number. ELSE, the total number of data in the list is an odd number. The calculation of the median is not done in the same way if the number of data is even or odd.

Review (15 minutes)



Have students walk around the classroom and look at other students' pseudocodes and codes.

- What do you notice?
- Are your peers' pseudocode the same as yours?
- Are your peers' code the same as yours?
- What would you change in your code? What suggestions would you give to your peers?
- What are the lessons learned today?

In order to continue learning in coding, continue to practice passing from a pseudocode to a code, or from a code to a pseudocode.

Ask students if there are other ways of coding to analyze data (for example, spreadsheets, such as Google Sheets and MS Excel).

Ask students to explain the benefit of creating code related to measures of central tendency in block-based coding software, instead of using =AVERAGE and =MEDIAN commands in a spreadsheet. Block-based coding gives us the opportunity to see what happens behind the scenes when using shortcuts in a spreadsheet. Creating this code allows us to explore the power of conditional blocks *(if, then, else)* and loops *(repeat, repeat until)*.

Consolidation of Learning

Provide students with an activity in which a table of values can be generated using a defined count, that is, a loop in which a list of values is generated according to the term number and the term.

Some examples of activities that use the "repeat until...":

- Create code that collects an exact number of data (for example, repeat until number of responses = 10) to facilitate the calculation of certain measures of central tendency. Here, the number could also correspond to the number of students in the class so that each student has the right to only one answer.
- Create code that collects data until a maximum sum is reached (for example, repeat until sum of values = 250). This kind of code could be useful for cumulative entry of points to hit a target. It would then be possible to use the measures of central tendency to determine the mean or the median of points for each entry.

CONSIDERATIONS

Links to Other Curriculum Expectations

Number

B2.1 Use the properties and order of operations, and the relationships between operations, to solve problems involving whole numbers, decimal numbers, fractions, ratios, rates, and percents, including those requiring multiple steps or multiple operations.

By using conditional statements *(if, then, else)*, it would be possible to create code that can analyze a mathematical problem and determine the processes to be followed to solve it by using subprograms (for example, subprograms for the priorities of operations, for the addition of fractions).

Algebra

C4. Apply the process of mathematical modelling to represent, analyse, make predictions, and provide insight into real-life situations.

Using subprograms and custom blocks allows you to generalize a situation by creating models and calling on those models at various points in the code.

Data

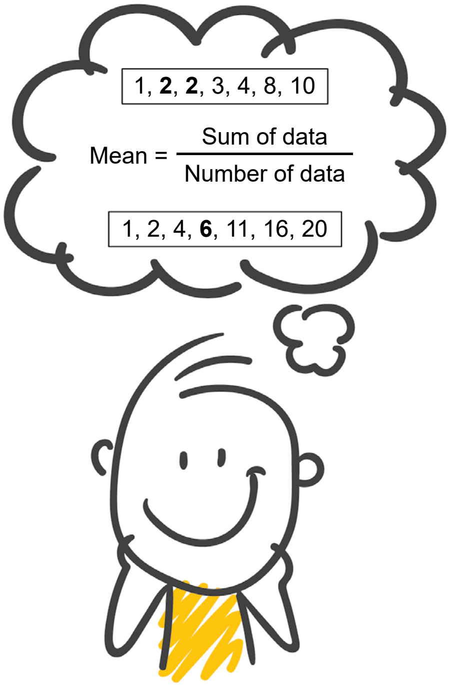
D1.5 Determine the impact of adding or removing data from a data set on a measure of central tendency, and describe how these changes alter the shape and distribution of the data.

By creating a subprogram for each measure of central tendency, it would be possible to write code that calls each subprogram at different times, creating more efficient and easy to read/alter code.

Differentiated Instruction and Universal Design for Learning

- Suggest to the student who is looking to take up a challenge that they create a code to find the mode of the students' shoe sizes. This code will use the list of data sorted in ascending order, but the conditions to be determined will be very different.
- By using coding software where project sharing is possible, some of the code could already be in place so that students do not always start from a blank canvas.
- Work at the pace of the students. Transferring mathematical knowledge and concepts to coding can take time. Allocate the necessary time to each part of the learning situation.
- Provide students with pseudocodes so they can turn them into codes. The pseudocode can also have a colour code that guides the student to the correct category of blocks in the chosen software.
- Provide a cheat sheet on the function of some more complex blocks, such as operators and conditions.

Appendix 1 – Warm-Up



Appendix 2 – Pseudocode

```
Variables: nb1
           nb2
           sum
Begin
    Set nb1 = 0.
    Set nb2 = 0.
    Set sum = 0.
    Define the value of nb1.
        Ask the question "What is the value of nb1?".
        The answer becomes the value of nb1.
    Define the value of nb2.
        Ask the question "What is the value of nb2?".
        The answer becomes the value of nb2.
    Sum = nb1 + nb2
End
```

Learning Situation – Grade 8

Title: We Love Veggies!
Duration: 150 minutes

Overview

The land next to the school, which belongs to the municipality, is for rent. The Grade 8 students decide to make a request to plant a community garden and sell their sprouts and produce at the market.

The students would like to raise enough money to pay for the costs associated with the purchase of hoodies for the entire student population of the school. The total cost would be $10 000.

Students need to use a spreadsheet to balance the budget taking into account fixed and variable expenses, as well as possible income. Students need to make predictions, make informed decisions and communicate them to the rest of the school.

Note: The project does not have to be a community garden. Question the students in order to find an entrepreneurial idea that interests them.

Overall and Spectific Expectations

Algebra

C3. Solve problems and create computational representations of mathematical situations using coding concepts and skills.

- **C3.1** Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves the analysis of data in order to inform and communicate decisions.

Financial Literacy

F1. Demonstrate the knowledge and skills needed to make informed financial decisions.

- **F1.2** Create a financial plan to reach a long-term financial goal, accounting for income, expenses, and tax implications.
- **F1.3** Identify different ways to maintain a balanced budget, and use appropriate tools to track all income and spending, for several different scenarios.

Preferred High-Impact Instructional Practices in Mathematics

Learning Goals, Success Criteria and Descriptive Feedback

Before creating the spreadsheet, the teacher determines with the help of the students the learning goals and the success criteria. This makes the intent of the activity more evident, so students can get a clearer picture of what is expected. Students can thus more easily create connections between their personal experience and other areas or subjects.

Tools and Representations

The use of tools in this activity helps students visualize a concept that is often very abstract, namely a budget. Using a tool such as an electronic spreadsheet helps the student see changes in the budget and how much money is missing before reaching their financial goal.

Small-Group Instruction

Teachers can use small-group instruction to more easily meet the needs of individual students. Students then feel more comfortable in new situations. Teachers are also able to reinforce understanding of the various functions employed during budget preparation.

Prior Knowledge and Skills

To be able to carry out this learning situation, the student must:

- master the addition, subtraction and multiplication of sums of money;
- have some understanding of measures of central tendency, especially the mean;
- know the interface of an electronic spreadsheet or software that can be used to create a spreadsheet;
- have certain knowledge specific to financial literacy such as the importance of balancing a budget and the different elements of a financial plan.

Learning Goals

At the end of this learning situation, the student will be able to:

- create a spreadsheet to effectively represent a balanced budget;
- use spreadsheet functions to automate certain calculations;
- represent expenses and income from the sale of plants and vegetables;
- create a plan to achieve a long-term financial goal of purchasing hoodies for all students in the school.

Criteria According to the Achievement Chart

Knowledge and Understanding

- The student represents a budget plan using an electronic spreadsheet.
- The student understands mathematical situations computationally.
- The student uses functions such as sum, mean and product to calculate expenses and income.

Thinking

- The student makes predictions and makes informed decisions using a chart representing their budget plan.
- The student creates a financial plan with a goal in mind.

Communication

- The student communicates clearly and effectively with peers to justify additions and adaptations to the project spreadsheet.
- The student creates representations using a spreadsheet and graphs.

Application

- Student makes connections to personal experiences to create a budget.
- The student uses functions such as sum, mean and product to calculate expenses and income.

Materials

- computer that has access to an electronic spreadsheet
- appendix (Warm-Up)

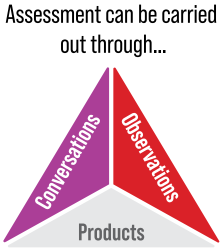
Note: The examples in this learning situation were made with Google Sheets.

Mathematical Vocabulary

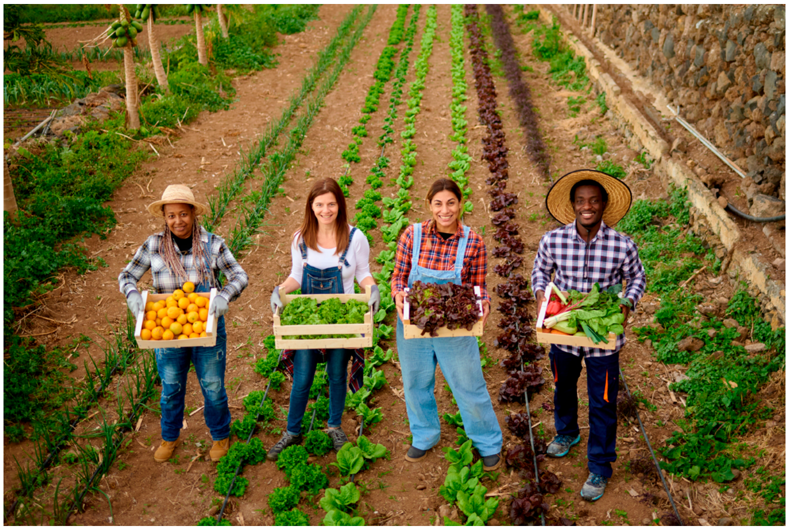
financial goal, income, expenses, balanced budget, fixed, variable (in the financial context), sum, product, mean

CONTENTS

Before Learning (Warm-Up) (30 minutes)



Show students the picture below (appendix) and ask them to formulate a mathematical question related to it. Students can also make observations about the picture.



Sought-after reflections

- How much money would the students earn if their vegetables were sold at the market?
- At what prices should their products be sold to make a profit?
- How do you properly account for garden sales and expenses?

Propose the development of a budget for the sale of products, the financial objective of which is the purchase of hoodies for the students of the school.

Have a discussion with students about the importance of a budget. Discuss the important components of a financial plan. Here are some examples of topics that students could focus on:

Cost

Monthly Expenses

Spreadsheet

Variable Expenses

Fixed Expenses

Long-Term
Financial Goal

Income

Balanced Budget

Possible student questions:

- What should be added to the spreadsheet, other than expenses and income?
- How will expenses and income be taken into account from month to month?
- Can we determine how much money we have left to raise to reach our goal?

Make a plan with the students to list the fixed and variable expenses to be in their budget. It would be important to count one-time expenses separately, which are not to be repeated each month, such as the purchase of seeds and soil.

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student does not know where to start.	Ask the student to make connections with their personal experience. • Have you ever been to the market? • Have you ever made a garden?
The student cannot identify the important elements of a budget.	What is the purpose of a budget? What are we trying to do in preparing a budget? Why is a budget important?
The student cannot differentiate a variable expense from a fixed expense.	Make connections to students' personal experience. • What expense do you and your parents have only a few times (variable expense)? • What expense do you and your parents have on a regular basis (fixed expense)?

Active Learning (Exploration) (60 minutes)

Example of budget made with Google Sheets: We Love Veggies!

Ask the students to write, on paper, a draft of their budget and to insert all the relevant information noted during Warm-Up. Then ask them to determine the calculations needed for the budget.

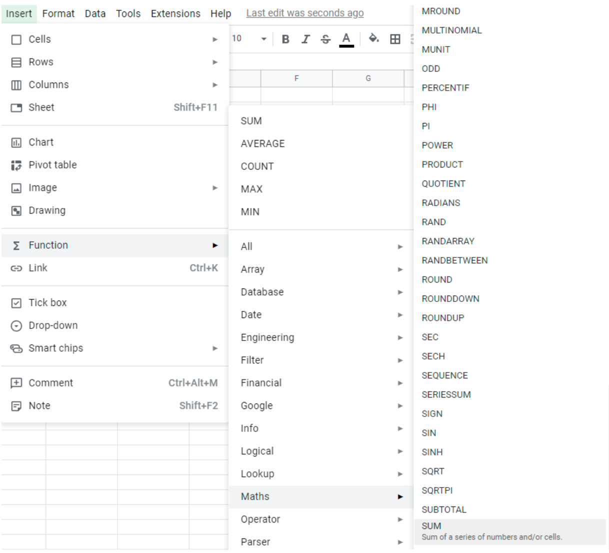
Possible Answers

- I have to add up all the expenses in order to know the amount of money that I spend initially and per month. (The "=SUM" code is used to sum a series of cells instead of adding them one at a time.)
- Determine the unit cost of the plants to be sold, then multiply it by the quantity. (In a spreadsheet, the asterisk (*) often represents multiplication. You can also use the code "=PRODUCT".)
- It is necessary to determine the difference between the money raised and the money spent (income – expenses).
- We need to determine the amount of money we have left to raise in order to reach our goal ($10 000 – total raised).
- I would like the cells to change colour to indicate potential issues; for example, if you spend more than you earn in a month. (Conditional formatting uses *if, then* language to change the properties of cells in a given area.)

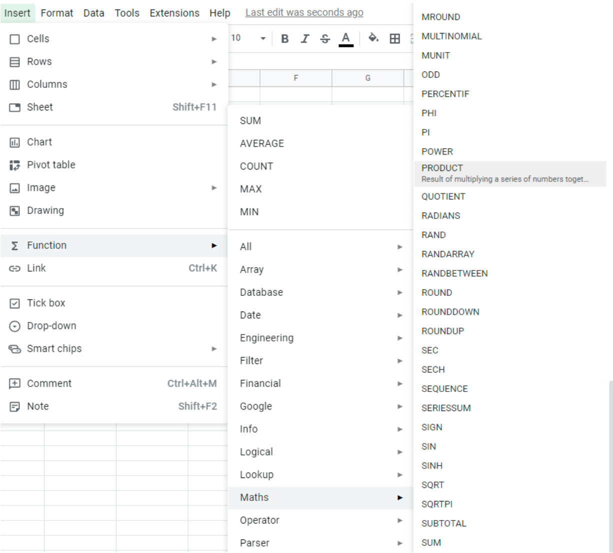
Ask students to identify the functions needed to create their spreadsheet.

Here are some examples of functions present in the Google Sheets spreadsheet. These functions remain accessible in different spreadsheets, but the exact place to find them in the menus may vary:

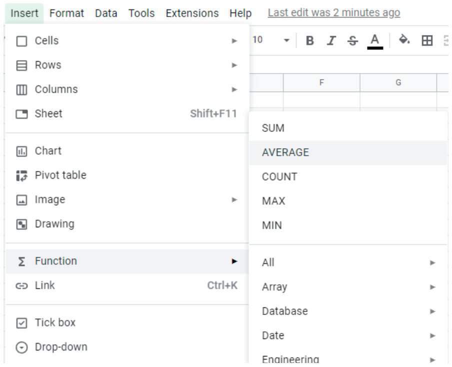
SUM



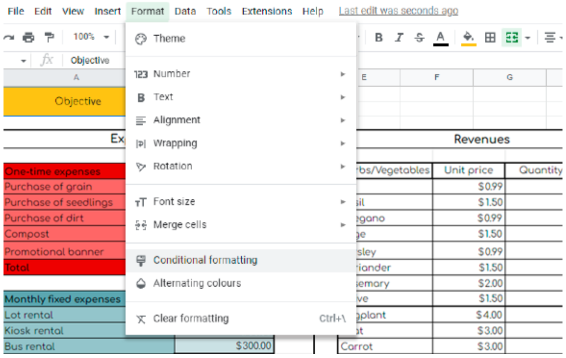
PRODUCT



AVERAGE



CONDITIONAL FORMATTING

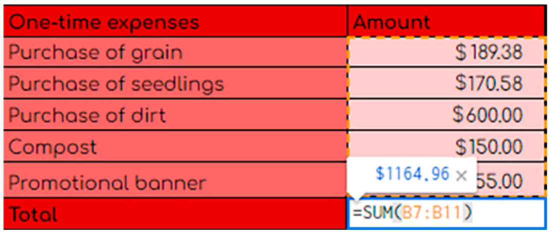


Note: The student can change the appearance of cells in the spreadsheet, depending on the data in them, using conditional formatting. This could be very useful in order to visualize the data in a budget, even if it is not a mathematical formula as such.

Challenge students to plan a budget for the community garden project. Encourage them to explore the software and try to apply the functions mentioned above.

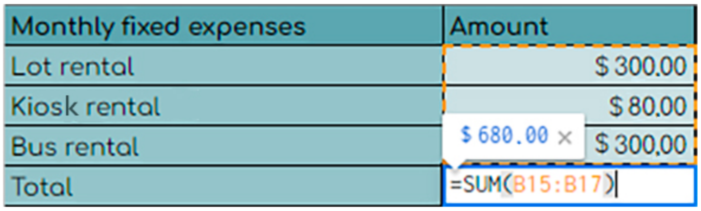
Examples of budget tables and functions:

Examples of one-time expenses:



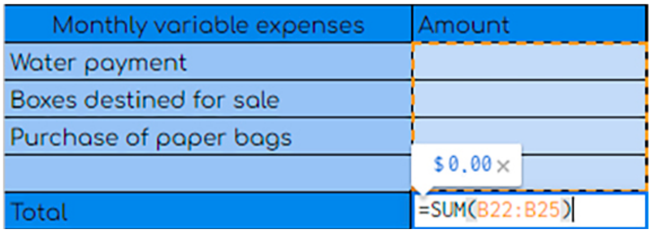
One-time expenses	Amount
Purchase of grain	$ 189.38
Purchase of seedlings	$170.58
Purchase of dirt	$600.00
Compost	$150.00
Promotional banner	$1164.96 × 55.00
Total	=SUM(B7:B11)

Examples of fixed monthly expenses:



Monthly fixed expenses	Amount
Lot rental	$ 300.00
Kiosk rental	$ 80.00
Bus rental	$ 680.00 × $ 300.00
Total	=SUM(B15:B17)

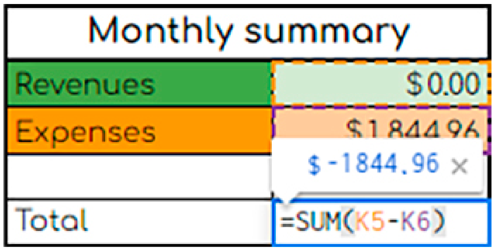
Examples of variable monthly expenses:



Monthly variable expenses	Amount
Water payment	
Boxes destined for sale	
Purchase of paper bags	
	$ 0.00 ×
Total	=SUM(B22:B25)

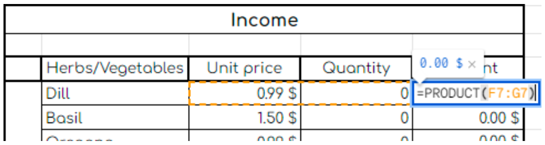
Note: The most used functions are SUM for addition and subtraction (when the negative sign is present for a datum, the software subtracts this datum in the "SUM" function). The PRODUCT function is also used when calculating income.

Example of SUM:



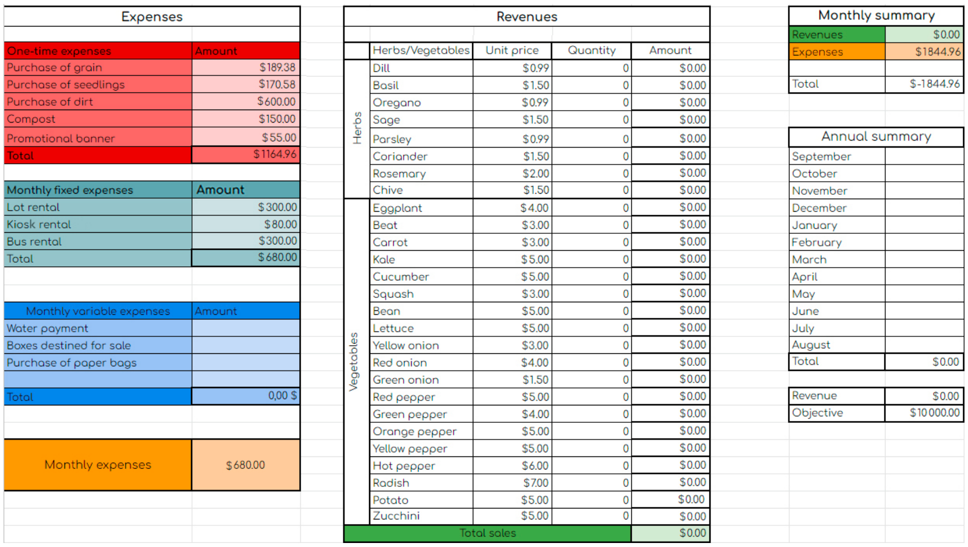
Monthly summary	
Revenues	$ 0.00
Expenses	$ 1 844.96
	$ -1844.96 ×
Total	=SUM(K5-K6)

Example of PRODUCT:



Income				
	Herbs/Vegetables	Unit price	Quantity	0.00 $ × nt
	Dill	0.99 $	0	=PRODUCT(F7:G7)
	Basil	1.50 $	0	0.00 $
	Oregano	0.99 $	0	0.00 $

Example of a blank document:



Expenses	
One-time expenses	Amount
Purchase of grain	$189.38
Purchase of seedlings	$170.58
Purchase of dirt	$600.00
Compost	$150.00
Promotional banner	$55.00
Total	$1164.96
Monthly fixed expenses	Amount
Lot rental	$300.00
Kiosk rental	$80.00
Bus rental	$300.00
Total	$680.00
Monthly variable expenses	Amount
Water payment	
Boxes destined for sale	
Purchase of paper bags	
Total	0,00 $
Monthly expenses	$680.00

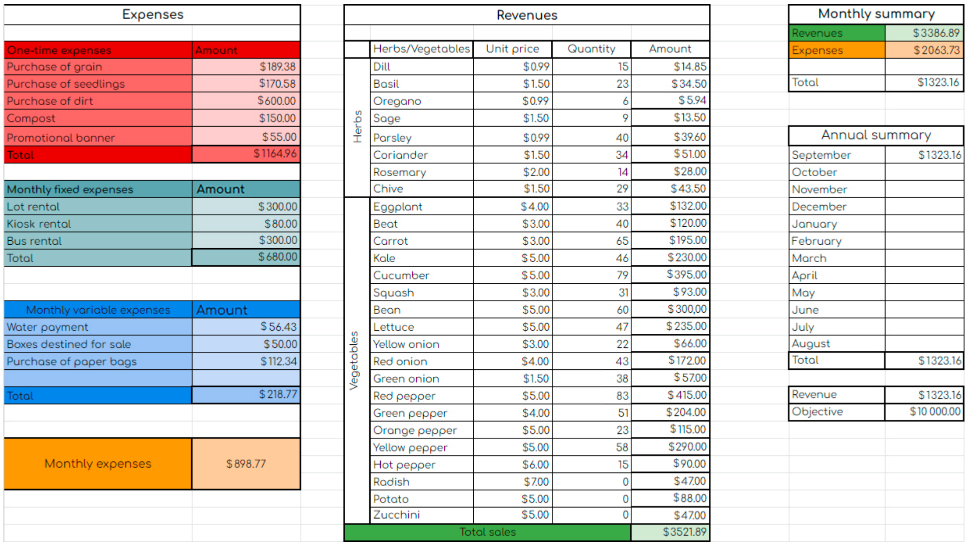
Revenues				
	Herbs/Vegetables	Unit price	Quantity	Amount
Herbs	Dill	$0.99	0	$0.00
	Basil	$1.50	0	$0.00
	Oregano	$0.99	0	$0.00
	Sage	$1.50	0	$0.00
	Parsley	$0.99	0	$0.00
	Coriander	$1.50	0	$0.00
	Rosemary	$2.00	0	$0.00
	Chive	$1.50	0	$0.00
Vegetables	Eggplant	$4.00	0	$0.00
	Beat	$3.00	0	$0.00
	Carrot	$3.00	0	$0.00
	Kale	$5.00	0	$0.00
	Cucumber	$5.00	0	$0.00
	Squash	$3.00	0	$0.00
	Bean	$5.00	0	$0.00
	Lettuce	$5.00	0	$0.00
	Yellow onion	$3.00	0	$0.00
	Red onion	$4.00	0	$0.00
	Green onion	$1.50	0	$0.00
	Red pepper	$5.00	0	$0.00
	Green pepper	$4.00	0	$0.00
	Orange pepper	$5.00	0	$0.00
	Yellow pepper	$5.00	0	$0.00
	Hot pepper	$6.00	0	$0.00
	Radish	$7.00	0	$0.00
	Potato	$5.00	0	$0.00
	Zucchini	$5.00	0	$0.00
Total sales				$0.00

Monthly summary	
Revenues	$0.00
Expenses	$1844.96
Total	$-1844.96

Annual summary	
September	
October	
November	
December	
January	
February	
March	
April	
May	
June	
July	
August	
Total	$0.00

Revenue	$0.00
Objective	$10 000.00

Example of document with data:



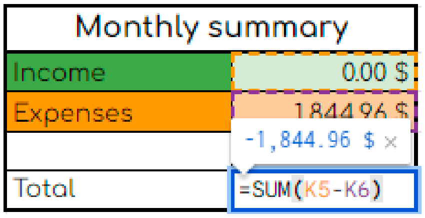
Expenses	
One-time expenses	Amount
Purchase of grain	$189.38
Purchase of seedlings	$170.58
Purchase of dirt	$600.00
Compost	$150.00
Promotional banner	$55.00
Total	$1164.96
Monthly fixed expenses	Amount
Lot rental	$300.00
Kiosk rental	$80.00
Bus rental	$300.00
Total	$680.00
Monthly variable expenses	Amount
Water payment	$56.43
Boxes destined for sale	$50.00
Purchase of paper bags	$112.34
Total	$218.77
Monthly expenses	$898.77

Revenues				
	Herbs/Vegetables	Unit price	Quantity	Amount
Herbs	Dill	$0.99	15	$14.85
	Basil	$1.50	23	$34.50
	Oregano	$0.99	6	$5.94
	Sage	$1.50	9	$13.50
	Parsley	$0.99	40	$39.60
	Coriander	$1.50	34	$51.00
	Rosemary	$2.00	14	$28.00
	Chive	$1.50	29	$43.50
Vegetables	Eggplant	$4.00	33	$132.00
	Beat	$3.00	40	$120.00
	Carrot	$3.00	65	$195.00
	Kale	$5.00	46	$230.00
	Cucumber	$5.00	79	$395.00
	Squash	$3.00	31	$93.00
	Bean	$5.00	60	$300,00
	Lettuce	$5.00	47	$235.00
	Yellow onion	$3.00	22	$66.00
	Red onion	$4.00	43	$172.00
	Green onion	$1.50	38	$57.00
	Red pepper	$5.00	83	$415.00
	Green pepper	$4.00	51	$204.00
	Orange pepper	$5.00	23	$115.00
	Yellow pepper	$5.00	58	$290.00
	Hot pepper	$6.00	15	$90.00
	Radish	$7.00	0	$47.00
	Potato	$5.00	0	$88.00
	Zucchini	$5.00	0	$47.00
Total sales				$3521.89

Monthly summary	
Revenues	$3386.89
Expenses	$2063.73
Total	$1323.16

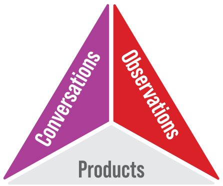
Annual summary	
September	$1323.16
October	
November	
December	
January	
February	
March	
April	
May	
June	
July	
August	
Total	$1323.16

Revenue	$1323.16
Objective	$10 000.00

POSSIBLE OBSERVATIONS	POSSIBLE INTERVENTIONS
The student does not use the "SUM" function.	Ask the student if there is a function noted at the beginning that could be used to simplify the task. Remind the student that using the spreadsheet simplifies the task of budgeting.
The student is unable to organize the spreadsheet effectively.	Ask the student to refer to the sketch at the beginning. Suggest that the student use colours to indicate the different expenses and the various incomes, then organize them.
Student fails to subtract.	Point out that you can subtract two cells by inserting a basic calculation.  Monthly summary Income 0.00 $ Expenses 1844.96 $ -1,844.96 $ × Total =SUM(K5-K6)

Review (60 minutes)

Assessment can be carried out through...



Group the students together and ask them to share the spreadsheet created with the other students in the class. Highlight the elements that are similar and those that are different.

Possible reflections:

- Some budgets are more or less realistic than others.
- My plan takes into account some expenses that other plans have omitted (the opposite reflection could also be relevant).
- By making a budget, we realize that starting a business can be complex!

Ask the students if it is possible to create a plan for the class by grouping together certain elements from all the plans.

- Is this new plan realistic?
- Will this new plan achieve the goal of raising $10 000?

Consolidation of Learning

Ask students to make a budget using a spreadsheet for a first-year apprenticeship, post-secondary, or labor market student. This spreadsheet should include expenses related to professional life or student life (tuition fees, apprenticeship fees, purchase of tools, transportation, training, etc.) and personal life (rent, bills, groceries, etc.), as well as possible sources of income (employment, scholarships, loans, donations, etc.). Students can use the garden budget plan as a starting point.

CONSIDERATIONS

Links to Other Curriculum Expectations

Data

D1.2 Collect continuous data to answer questions of interest involving two variables, and organize the data sets as appropriate in a table of values.

It would be possible to collect data, either manually or with survey software, then transfer the data to a spreadsheet for analysis and interpretation through code.

Spatial Sense

E2.4 Describe the Pythagorean relationship using various geometric models, and apply the theorem to solve problems involving an unknown side length for a given right triangle.

It would be possible to create code to draw right-angle triangles using a table of values that gives two of the three side lengths. Alternatively, the student could create code that generates a table of values related to the measured lengths of sides *a* and *b* and then generates the length of the hypotenuse.

Financial Literacy

F1.4 Determine the growth of simple and compound interest at various rates using digital tools, and explain the impact interest has on long-term financial planning.

Using simple and compound interest operations, students will be able to generate lists of investment or loan values over time to compare them.

Differentiated Instruction and Universal Design for Learning

- Provide students with a sample budget. The example could be completely blank to give them an idea of how a budget should be structured. It could also include some data aimed at activating students' knowledge.
- Invite students to work in teams, either to encourage peer learning or to foster direct and personalized instruction according to their needs.
- Create repositories of the various mathematical functions under study and ways to insert them into the spreadsheet.
- Present certain data in the form of graphical representations. Ask them to explore graphing tools to find visual representations that might be useful and relevant.

Appendix – Warm-Up

4. AND THEN...

The Student's Experience

Connections Between Coding and STEM (Modern Problems Require Modern Solutions)

In order to provide students with a mathematical experience, it is important to take into account the omnipresence of technology in our society and its constant evolution. In the classroom, several subjects can be integrated into the mathematics curriculum in order to contextualize them and make them more accessible and relevant.

STEM instruction aims for interdisciplinary learning of science, technology, engineering and mathematics. Since it is difficult, if not impossible, to apply STEM disciplines to solving societal problems without considering the aesthetic and artistic aspect of the proposed solutions, the letter "a", for arts, is added to the acronym, namely STEAM. By integrating the different STEM disciplines into instruction, students evolve in authentic contexts to find solutions to real and modern problems.

Coding can be used as a starting point and thread for STEM instruction in the classroom. Since coding is not a strand in itself, but rather an approach to interpreting, representing, and solving a problem, it can be used to make connections between mathematical strands and other subjects. This integration of targeted mathematical concepts into another curriculum area, such as science and technology, contextualizes mathematics to make learning the subject more accessible to students and to assess multiple knowledge and skills at the same time. Here are some examples of how coding can been integrated into STEM disciplines.

S - Science

Science includes several subfields, such as biology, chemistry, physics, astronomy, and earth science. The advancements made in all these areas have been possible thanks in particular to robotics and coding. Students may not be aware of the importance of coding and robotics in science. Satellites in space are a prime example. Since satellites are unmanned, several important processes must be automated so that these spacecraft can travel exceptional distances or maintain an orbit around a celestial body.

In the classroom, students can use coding software to draw a round shape around an object, as if it were an orbiting satellite. For a greater challenge, the object around which the satellite orbits could be in motion. Students could also create code to program a robot that would collect data, such as weather data, and return it for analysis. Robots, like LEGO Mindstorm, LEGO WeDo and Dash and Dot, or microcontrollers, like micro:bit and Arduino, could serve as sensors for this purpose.

Tools used in medicine, such as heart rate monitors or blood glucose meters used by people with diabetes, are another example of coding in science. These tools are essential to the survival of many people and work from conditional statements.

In the classroom, students can create their own code to collect heart rate data using microcontrollers, such as Arduino, micro:bit, and Grove Zero. Students can then use a spreadsheet to enter their data and represent it using a variety of graphs.

T – Technology

Technology is the STEM discipline that is most often and easily associated with coding. It is used in many workplaces, including agriculture and medicine.

Farmers use technology to efficiently operate large agricultural lands. Industrial irrigation and seed systems are often partially automated. They use sensors to determine temperature, moisture and soil chemistry. The sensors also allow for self-guidance of farm machinery and the use of GPS to ensure proper positioning of the systems.

In the classroom, students can run simulations of irrigation systems, such as those used by farmers, or create a miniature version of an irrigation device using Arduino or other microcontrollers.

Motorized wheelchairs are an asset to people with limited mobility because they provide them with a degree of independence. These motorized wheelchairs could not exist without the technology of manufacturing and programming. There are several inputs, such as the movement of a joystick, and several outputs, such as movement in the desired direction, that must be programmed. The systems are rigorously tested to ensure the safety of the user.

In the classroom, students can explore an accessibility problem using automation and robotics.

E - Engineering

Engineering is a field that includes a multitude of elements. It consists of "the study of an industrial project in all its aspects (technical, economic, financial, monetary and social), which requires a synthesis work coordinating the work of several teams of specialists[1] ". Several aspects of engineering work can be facilitated with the help of coding.

Block-based or text-based coding is also a form of engineering. The coder creates structures that interact efficiently and harmoniously. The coder must test the code in several scenarios to ensure that it works properly before making the final product available to the target audience. This process is very similar to the engineering process associated with designing a structure, such as a building or a bridge, or designing a machine. There is an area of electrical engineering called computer engineering, in which engineers develop computer devices and the codes that make them work.

[1] Larousse.fr. (nd). French dictionary. [Online]

Engineers can use coding to simplify their work and be more efficient. Repetitive mechanical processes can be automated using loops, and design software can simulate the strength of a structure by applying conditional statements. These are just two simplified examples of the use of coding in an engineering context.

In the classroom, students can use coding in a robotics context to automate a process; for example, a mechanical arm that places an object on an assembly line. Students can also create codes to automate the calculations that determine the costs of an engineering project using a spreadsheet and mathematical formulas.

M – Mathematics

Coding cannot exist without mathematics. Whether it is block-based or text-based coding, the creation of code depends on a multitude of calculations and logic problems. Loops, for example, are just repeated addition or multiplication, and conditional statements depend on equalities and inequalities to know how to proceed.

Coding, in mathematics, is often used in the area of finance, for example, in the use of software that uses functions and conditions to account for expenses and incomes and to balance budgets. Conditional statements are used to quickly visualize the balance of the budget or calculations (for example, by highlighting data that matches the programmed conditions).

In the classroom, students can also quickly create a budget and ensure it is balanced using functions and conditional statements.

Coding is also used in spatial reasoning contexts. As automation grows in importance in our world, it is important to have people who are good at coding who will create programs that allow robots to make deliveries or cars to drive autonomously. These people must take into account distances, geographic components, such as relief in the land, weather conditions and physical components, such as roads and traffic lights, to plan the most efficient and safe route possible so that robots can perform the desired movements.

In the classroom, students can program a robot to follow a particular path or to find the most efficient route to get to a given point by using electronic maps and having the robot move along the planned route. Programmable robots such as the LEGO Mindstorm, LEGO WeDo, Bee-bot, Blu-bot, Ozobots or Dash and Dot can all be useful. Some of the platforms also have online simulators.

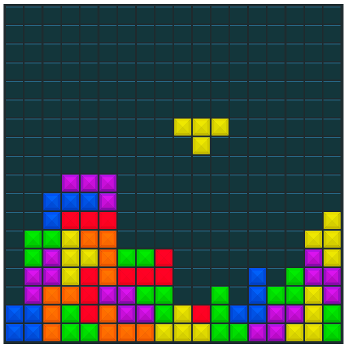
Connections Between Coding and Real Life

Connecting mathematics to the student's life, including their interests and experiences, can not only help make concepts that are sometimes very abstract concrete, but can also foster a desire to learn in the student. The link between coding and reality exists even before the student realizes it because of the ubiquity of computers and the speed with which technology develops.

Video games are a good way to connect a student's life to coding. The code that makes a video game work is very mathematical.

Here are some examples of connections between video games, coding and mathematics:

- In order to move a character, be it an avatar, a vehicle or a block, like the original version of the game *Tetris®*, it is necessary to code instructions to make the character act according to the inputs, either the mouse, the keyboard or the joystick. The character is in a Cartesian plane, which makes it possible to know its exact position. Thus, it is possible to make it interact with other characters or non-player characters (NPC).



- The statistics associated with the different avatars are a form of mathematical modelling. The person writing the code creates a basic avatar with editable stats. Then, using this avatar, they create all the others. In a car racing game, for example, the cars all have different speeds and abilities. By creating a single master code that includes the ability to change these two parameters, it makes programming much easier.

These are just two examples of an unlimited number of possibilities. It is therefore important to ask students questions to make them aware of the mathematical elements involved in video games.

The video game is an effective common thread between the student's life, coding and mathematics, but it must be remembered that not everyone plays video games. It is therefore important to offer them other examples related to coding.

Here are a few:

- The barcode that determines the price of an object requires coding. The distances between the bars on the barcode are accurately measured to determine product identity. The price of the product is programmed into the store's checkout system. It only takes a few seconds, but complex math goes into every purchase. 2D barcodes (QR codes) work in a similar way, but can contain much more information.

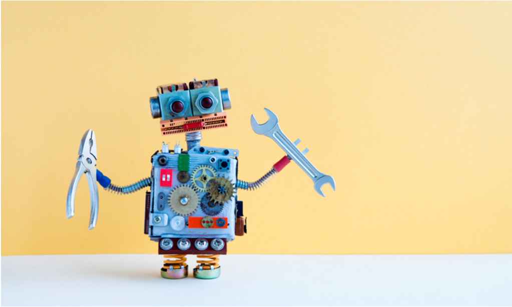
  This connection could also be a gateway to financial literacy, especially related to responsible consumption, such as the creation of budgets as well as the advantages and disadvantages of various payment methods.

- Websites and online media, whether social media or TV and movie streaming services, use complex algorithms in order to make recommendations to the user. Platforms offering movies and series, for example, use code to process data related to viewings. They create lists of movies watched, which are then sorted. The code determines commonalities and generates movie suggestions. It's the same operation for websites or social media.

  Students who visit websites, for example, to learn more about famous personalities, and who use online media, as well as students who aspire to create content for the web, need to be familiar with how these algorithms work.

- Modern household appliances also work with code. The washing machine, dryer, and dishwasher all have codes that map the buttons on the machine to the controls they are linked to. Temperature and humidity sensors, for example, collect data. Then, conditional instructions are used to ensure that the correct temperature is reached. It's the same with the refrigerator, freezer, microwave, etc. In addition, some appliances can connect to a wireless Internet network. This connection requires a code in order to send and receive information over the network.

Students may not have a particular interest in home appliances, but this connection can be a great way to explore robotics. The world of robotics is much more than the creation of robots. The goal of robotics is to use technology and coding to solve problems and make processes more efficient.



The connections between coding and a student's daily life are numerous, if not limitless. The examples presented are just a sampling. In order to allow students to take full advantage of the integration of coding into the mathematics classroom, it is important to allow them time to explore and discover how coding fits into their lives.

Connections Between Coding and Careers

It is impossible to predict the labor market of the future. Today's careers may disappear or evolve to such an extent that they are no longer recognizable. For this reason, it is important to equip students not to target specific careers, but rather to meet challenges. This approach allows for greater flexibility in terms of career path. Coding plays an important role in this approach by developing a positive math-related identity.

Coding requires learning a "language", that is computer language. Indeed, it is necessary to assemble pieces, whether blocks or text, in order to create a whole, a bit like creating a sentence with words or a paragraph using sentences. In addition, code sharing requires the writing of comments explaining the decisions made at the time of programming. Coding therefore helps to develop language and communication skills, which are used with the computer, but also with other students. This ability to communicate in various contexts and for different audiences will be an asset, regardless of the job the student holds in the labor market.

Technology is changing rapidly. More and more tasks and processes are computerized. Knowing how to code offers several advantages in such a reality. Coding goes beyond learning a new language, it gives the student a new way of representing a problem and solving it. This flexibility of thought becomes an asset when the student tries to meet a challenge. Coding develops student resilience, critical thinking and creativity.

The Canadian Occupational Projection System (COPS), in its report containing its career predictions for the years 2019 to 2028, explains that automation and the advancement of technology will create a significant number of jobs, even if these processes will make some jobs obsolete. In addition, a report published by Deloitte entitled The Intelligence Revolution points out that humans and computers have different but complementary skills. These findings show the importance of allowing students to familiarize themselves with the computer, as a tool, and also with all the processes related to the operation of a computer when it performs a task. Everything the computer can do is made possible by running code. An understanding of how coding works allows the student to better understand the devices used in everyday life.

The knowledge and skills that students acquire through coding give them an advantage in their post-secondary studies, in an apprenticeship or when entering the job market. Coding is becoming a valuable skill in science, math, engineering, and computer science, among others. Coding skills help, among other things, to strengthen problem solving, develop a collaborative spirit, stimulate curiosity and encourage questioning. Problem solving is common and frequent in today's and tomorrow's world. Having coding skills solves challenges in multiple job fields and many aspects of a job.

Coding Terminology

TERM	SHORT DESCRIPTION
ACTUATOR	Allows you to perform actions in an automated system. A motor is an actuator.
ANALOGUE PIN	A pin that can send or receive a range of numeric values.
ARGUMENT	The value given to the variable when calling a function (subprogram).
ARRAY	A data structure that holds multiple items, which can also be known as a list.
BOOLEAN	A value that can be either true or false. In block-based languages, Boolean values are often represented by a hexagon.
CHARACTER	Sign used to form words and text, including upper and lower case letters of the alphabet and numbers.
COMMENT	Code ignored by the machine, but which allows humans to add annotations.
COMPARISON	Allows you to compare two values or two expressions; for example, >, <, =, ≤, ≥.
CONTROL STRUCTURES	Structure of the code that allows the flow of the program to be changed (for example, loops, conditional statements).

TERM	SHORT DESCRIPTION
COORDINATES	Position of an object in a plane expressed using an *x*-value (horizontal) and a *y*-value (vertical) or in a 3-dimensional space (*x, y* and *z*). Note that some coding interfaces use the four quadrants of the Cartesian plane that the student knows, while other interfaces use the upper left corner as the point (0, 0), with values increasing in *x* as they go to the right and increasing in *y* as they go down.
DEBUGGING	Find and eliminate errors in the code.
DECLARE	Allows you to introduce a new variable.
DEFINE	After having introduced a variable, allows to give it a value.
DIGITAL PIN	A pin that can send or receive only two values, usually ON or OFF (or HIGH or LOW).
DISCRETE	These are data that are distinct or discontinuous.
EVENT	Allows you to trigger part of the code; for example, when clicking on the flag, on a button, when the sound intensity exceeds a certain value.
EVERY...	A loop that repeats at a certain interval; for example, every 5 seconds.
FOR... FROM 0 to...	Loop repeating a certain number of times.
FOREVER	A loop that repeats indefinitely.
FUNCTION	Subprogram that can return a value.
FUNCTION CALL	The instructions execute the subprogram.
IF... THEN... ELSE...	Used to create a conditional structure in which there are two options.
INCREMENT	Repeatedly adding a fixed value to a variable. Note: It is possible to add a negative value to decrease the value.
INDEX	A variable used to locate the position of an element in a list.
INFINITE RECURSION	When a function calls itself, an infinite loop is created.
INPUT	Enter data into a computer or microcontroller.
INTEGER	A number that is neither a fraction nor a decimal number (this includes positive and negative integers). A variable can be declared to contain only integers.

TERM	SHORT DESCRIPTION
JOIN	Allows joining several character strings end to end. In block-based languages, we often speak of "joining" or "grouping".
LED	Light-emitting diode: an electronic component that produces light.
LEN	Short for "length", allows you to determine the length of a list.
LIST	Variable containing more than one value often called a one-dimensional array.
LOCAL VARIABLE	Variable that is only accessible within a function or *sprite*.
LOGIC DIAGRAM	Diagram that shows the structure of a process.
LOOP	Allows you to repeat instructions.
MAP	Converts a value from one scale to another; for example, if a sensor gives values between 168 and 875, the values can be reduced to a scale of 0 to 100.
MAX	The greatest value.
MICROCONTROLLER	Small electronic circuit to control a device (for example, Micro:bit, Arduino, LEGO Hub).
MIN	The smallest value.
MOD or REMAINDER	The remainder of a division (for example, 27 MOD 5 is 2).
NUMERICAL VALUE	A value that is a number. In block-based languages, numerical values are often represented by a rounded shape.
OBJECT	Basic element that allows you to code. *Sprites* are objects. Objects have an identity, properties, behaviours and code attached to them.
ON START	Contains the code to be executed when the microcontroller starts up.
OPERAND	Data or value entering an arithmetic operation. In block-based languages, there are often holes in the operator blocks to insert the operands.
OPERATOR	Symbol that represents an arithmetic (for example, +, -, *, /) or logical (for example, AND, OR, &gt;, &lt;, =) operation.
OUTPUT	Sending information to the outside of the microcontroller.
PARAMETER	A variable that is passed to a function (subprogram).

TERM	SHORT DESCRIPTION
PIN	A small metal rod that connects an electronic component to a microcontroller. The pins are numbered, which makes it possible to choose one in the code.
POINTER	On-screen symbol often in the form of an arrow. In the code, we can create conditional structures "when the pointer touches... so...".
PRINT	Displays characters on the screen.
PSEUDOCODE	Description of the algorithm in everyday language before converting it into machine-understandable code.
RANDOM	A randomly determined value.
RANGE	Interval between two values.
ROUND	Rounds a numeric value.
SENSOR	Component used to collect data such as a temperature or light sensor.
STRING	Several characters grouped together; for example, words, phrases or a password.
TRUNCATE	Removing the decimal part (without using the rules for rounding).
VARIABLE	Can take on different values while the code is running.
WAIT	Allows you to insert a waiting time in the code. The pause is often expressed in seconds or milliseconds.
WAIT UNTIL...	Executes code until a condition becomes true. This is the opposite of the "while" loop.... ".
WHILE...	A loop that runs as long as a condition is true. This is the opposite of a "wait until...".

APPENDIX – MATHEMATICAL MODELLING

UNDERSTAND THE PROBLEM

- Invite students to formulate questions related to the problem to be solved.
- Facilitate a discussion so that the students discuss the questions formulated. Make sure, during the discussions, that the questions proposed are related to the concepts under study.
- In small groups, invite students to evaluate the questions and make thoughtful decisions in order to prioritize them.
- Ask each team to choose a key question for which a model will be developed to answer.
- Ask students the questions "What information do we already have?" and "What information do we need to solve the problem of the targeted question?".
- Invite the students to determine all the necessary elements to solve the problem (for example, list of materials, price, time required).
- Keep in mind that groups or students can work on different problems related to the same situation.
- Tell students that there are several ways to solve the problem. Invite them to identify missing data by doing research.

Observation	Lines of Questioning and Possible Interventions
The student has difficulty asking questions related to the targeted expectation.	• What do you want to know? (MP) • What do you understand about the problem? (MP) • What emotions are you currently feeling? Can you explain or draw the reason why you feel this way? (SELS) • How can you use critical thinking to make connections between this picture and the concept being studied? (SELS) Invite the student to review the analysis of the situation component in order to identify more facts and suppositions.
The students cannot agree on the prioritization of the questions.	• What words or clues in the questions are important in determining priorities? (MP) • What strategies can you use? (MP) • How can you be inclusive in order to respect the opinions of team members? (SELS) • How can you express your feelings and show that you understand the feelings of others? (SELS) Allow students to consider solutions such as flexible groupings or assigning different tasks within the team.
The student chooses a question that cannot be answered by a mathematical model (for example, "What flowers smell nice?").	• What complex question related to the concept under study do you ask yourself when looking at this picture? (MP) • What mathematical concept(s) is this question related to? (MP) • Can you name a challenge related to the question you developed? What lesson do you learn from this mistake? (SELS) • How does this situation allow you to develop your perseverance and resilience? (SELS)

ANALYSE THE SITUATION

- Show the students the picture representing the learning situation, then ask them the questions "What do you notice? and "What assumptions are you making?".
- Use various communication strategies such as the Think-Pair-Share strategy to answer questions.
- Invite students to record their observations individually. Then, group the students into teams and allow them to react to the questions by talking to each other.
- Come back in the large group to discuss the findings, thoughts and ideas exchanged during the team conversation. List relevant facts and assumptions about the situation.
- Have students review relevant facts and reasonable assumptions to determine the problem(s) to be solved.
- Following the observations, intervene with the students using a strategy related to the demonstrated need (SELS).

Observation	Lines of Questioning and Possible Interventions
The student seems to have difficulty making connections to personal experience and participates little in the exchange.	• Where have you ever seen this? (MP) • Have you ever talked about this in a science class? health? social studies? (MP) • Can you name the reason(s) why you feel this way? (SELS) • Did you explain your ideas to your partners, even if they were different? (SELS)
The student makes connections to personal experience, but these do not involve mathematical concepts.	• What does this problem make you think about? (MP) • What are the similarities or differences between the connections you make and mathematics? (MP) • Where would you place your emotion on an intensity thermometer? (SELS) • It is important to remember that one should not try to understand everything the first time. What could be the next step? (SELS)
The student fails to identify relevant facts or make reasonable assumptions about the concept under study.	• Visualize context from the image. What important information should be considered before even determining a problem? (MP) • What do you want to know about mathematics in this image? (MP) • Are you able to use an effective strategy to manage your emotions as you think about this situation? (SELS) • How can you use creative thinking to make connections between this image and math? (SELS)
The student easily identifies a problem and demonstrates an eagerness to move on to the next step.	• What were your observations compared to the initial image? (MP) • How do you feel about the time taken to analyse the situation? (SELS) If the student shows that a thorough analysis has been carried out and that the problem is related to it, allow them to proceed to the component of understanding the problem.

CREATE A MATHEMATICAL MODEL

- Remind students to consider all of the elements needed to solve the problem, identified at the time of understanding, while recognizing that some elements may be added as the model develops.
- Allow students time to work, think, and determine how to solve the problem by making several attempts and using different data, mathematical concepts, and skills to create a model.
- Ensure that students assume assigned roles in the team work to give them a sense of ownership in the task.
- Observe teams as they work and identify those who are having difficulty.
- Anticipate some challenges that students might encounter.

Observation	Lines of Questioning and Possible Interventions
The student has difficulty getting organized to create the model.	• How can you represent the data to help you better understand the problem? (MP) • What would be the steps to follow in order to create a model? (MP) • What would be a good strategy to solve this problem? (MP) • How can you use critical thinking to break down the task? (SELS)
The student forgets to include some important elements in their model.	Encourage the student to review the component of understanding the problem in order to bring out the elements necessary to solve the problem. (MP) • What do you need to determine? (MP) • What strategies can you use to continue building your model? (SELS)
The student has difficulty in developing a mathematical model related to their question.	• It is important to remember that one should not try to understand everything at once. What could be the next step? (SELS) • What questions could you ask your team members to better understand the problem to be solved? (SELS and MP) Provide examples of concrete, semi-concrete and symbolic models.
The student creates a model that solves the problem in a short time.	• Can you move on to the next step of the problem? (MP) • Can you use another model to answer the problem? (MP) Invite the student to find strategies to match the pace of work of peers. (SELS) Invite the student to provide support to the other teams, if necessary. (SELS)

ANALYSE AND ASSESS THE MODEL

- Encourage the student to analyse all the elements of their model in order to ensure that they correctly answer the different parts of the problem.
- Have the student evaluate their model.
- Collect all math models to display in the classroom with the targeted question.
- Ask students to walk around the classroom and observe all the models displayed.
- Ask students to comment and ask questions about the different models, while ensuring that the questions and comments are constructive and related to the instructional intent of the learning situation. Get students thinking by asking questions, such as:
  - Compare your work with that of other teams. Are you convinced of your solution? If yes, explain why. If not, modify your solution.
  - What are the similarities and differences between the different models used?
  - Is there a connection between the different models? If yes, which ones?

Observation	Lines of Questioning and Possible Interventions
The student fails to identify the similarities and differences between the various models.	• What did you notice? (MP) • What elements stand out in the mathematical models? (MP) • What do you think of how another team did it? (MP) • How can you be inclusive in order to respect the opinions of others? (SELS)
The student can compare their model with others and identify gaps.	• Do you have decisions to make? • What are the next steps? (MP) • If you had to answer the question again, would you use the same model? (MP) • Why do you think the model your friend used was more effective? (MP) • What lesson do you learn from this mistake? (SELS)
The student does not understand their mistakes or does not think in order to understand them.	• Does your solution make sense? Why? (MP) • Can you represent the idea differently? (MP) • What emotions are you currently feeling? (SELS)
The student's model could not be applied to everyday life.	• How can you use critical thinking to determine the plausibility of your model? (SELS) Invite the student to show their model to someone outside the classroom (for example, a staff member, another student in the school, a member of their family or community).
The model does not provide an adequate solution to the problem.	Encourage the student to revisit a previous component and redefine the problem, make new assumptions, or make changes to the model.

MP: Mathematical Processes
SELS: Socio-Emotional Learning Skills