SCIENCE NORTH SCIENCE NORD
SUDBURY, ONTARIO, CANADA

Grow with Code	Grade 3-4

Lesson Plan

Coding Tool	Scratch (or ScratchJr)
Time Required	Two periods

Math Curriculum Connections

Grade 3 and 4

Algebra: Coding

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

Specific Expectations

C3.1 solve problems and create computational representations of mathematical situations by writing and executing efficient 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 outcomes

Science Curriculum Connections

Grade 3

Life Systems: Growth and Changes in Plants

3. demonstrate an understanding that plants grow and change and have distinct characteristics

Specific Expectations

3.1 describe the basic needs of plants, including air, water, light, warmth, and space

3.4 describe how most plants get energy to live directly from the sun

Grade 4

Life Systems: Habitats and Communities

3. demonstrate an understanding of habitats and communities and the relationships among the plants and animals that live in them

Specific Expectations

3.1 demonstrate an understanding of habitats as areas that provide plants and animals with the necessities of life (e.g., food, water, air, space, and light)

3.3 identify factors (e.g., availability of water or food, amount of light, type of weather) that affect the ability of plants and animals to survive in a specific habitat

Description

What do plants need to grow? Using hands-on activities combined with on-screen coding activities, students will explore plants' needs and represent them using coding concepts. Students will practice using loops to build a program using Scratch to explore repeating patterns and concurrent events in the context of growing plants.

Success Criteria

By the end of this lesson, students should be able to describe and assign events as either sequential or concurrent. They should be able to use coding controls such as loops to describe repeating events, and explain, using coding algorithms, programs that include sequential, concurrent, nested, and repeated events.

Materials and Media

- Computer or iPad with access to Scratch (or Scratch Jr if using Coding Activity Variation)
- Grow with Code Handout
- Grow with Code Coding Guide or Grow With Code Variation – Scratch Jr Coding Guide for Beginners

Computational Thinking Skills

This lesson uses both a hands-on unplugged activity and an online block-based coding exercise to reinforce the concepts of algorithmic thinking and loops.

In the unplugged activity, students will explore the logic of sequential, concurrent, and nested events to see how rearranging the set of instructions in code (the algorithm) affects the output. They will also practice using loops to repeat events and make code more efficient. The **Grow with Code Handout** provides exercises that allow students to practice applying these concepts to instructions relating to plants and plant growth.

In the online activity, students will be able to combine coding concepts to examples of factors that affect plant growth in the environment. They will build and modify code using sequential, concurrent, and nested events, as well as control structures such as loops and conditional statements, to describe how the appearance of a plant sprite (output) changes, when it interacts with other graphical elements in the program.

The **Grow with Code Coding Guide** document for this lesson includes a detailed step-by-step procedure for using Scratch.

For a simplified version of this activity for beginner coders, there is also a step-by-step **Grow with Code Coding Guide for Beginners**, which uses Scratch Jr., a free block-based coding application with illustrated blocks in place of text.

Introduction

Introduction to Algorithmic Thinking

In coding, an algorithm is a set of steps (or instructions) that tells a computer program how to accomplish a task. Using good algorithms — writing good instructions — lets you create interesting and important programs.

A computer program cannot interpret instructions and make their own decisions like humans do. They can only follow instructions exactly as they are written. They can't add steps or change their order. So, it's important that, when you plan your code, you arrange your

instructions in a way that is clear, specific, and following the correct sequence.

Sequential events are steps that occur one right after the other. A computer program will start at the first step in the sequence, continue to the second step, then the third, and keep going in order until the final step is reached.

Concurrent events are steps that occur at the same time or during overlapping periods of time. In this lesson, we will be describing concurrent code in both the unplugged activity and in the Scratch coding activity by creating separate programs start at the same time.

Repeated events are described in code using loops. The set of instruction contained within a loop will repeat until a condition is met to tell the computer program to exit the loop and move on to the next steps of the algorithm. The condition of the loop might be to repeat forever, to repeat for a set number of cycles, or until an event occurs (e.g., a game might loop until all of the player lives are used up, at which point the program will exit the loop to trigger a "Game Over" screen).

Nested events are control structures, like loops and conditional statements, that are placed inside other control structures. An example of this would be a loop nested inside a loop. Nesting is useful for simplifying a program and making the code more efficient. In this lesson students will explore loops nested inside of loops during the unplugged activity, and conditional statements nested inside loops in the online coding activity.

Pseudocode is the process of writing out code offline, to help plan for what will be input with the computer later. Using pseudocode helps make code more efficient as it eliminates potential errors before investing too much time into the programming language. Students will practice planning their code by writing out instructions, incorporating control structures such as loops to describe repeated events, and clearly identify concurrent sets of instructions.

Introduction to Growth and Changes in Plants

We can describe plant growth as a sequence or series of steps with a specific order. For example:

1. A seed is planted and watered.
2. The seed germinates and becomes a seedling with roots and a sprout.
3. The seedling grows into an adult plant with a stem and leaves.
4. The adult plant produces flowers.
5. The flowers are pollinated by insects, animals, and with help from the wind.
6. The pollinated flowers produce fruit.

We can also describe plant growth using loops, since the steps involved in plant growth repeat

to form a cycle. For example:

1. A seed is planted and watered.
2. The seed germinates and becomes a seedling with roots and a sprout.
3. The seedling grows into an adult plant with a stem and leaves.
4. The adult plant produces flowers.
5. The flowers are pollinated by insects, animals, and with help from the wind.
6. The pollinated flowers produce fruit.
7. The fruit contain seeds that are spread by animals and dropped fruit.
8. Return to step one.

In this lesson, students will practice building and modifying algorithms, and they will apply this learning to describe the steps involved in plant growth.

Action

Unplugged Activity

The goal of this unplugged activity is to introduce the concepts of sequential versus concurrent events and the use of loops to repeat events. The activity will use a form of pseudocode to describe code as sets of instructions.

Part one: Sequential Steps

In coding, an algorithm is a series of instructions, or steps. Programs will read and execute your code in the order that you write it — so your order, or sequence, is important!

Let's look at a simple set of instructions.

Write the following instructions on the board and draw a box around the steps:

1. Stand up
2. Clap your hands
3. Clap your hands
4. Clap your hands
5. Sit Down

What are these instructions telling us to do? How would you follow these instructions?

Demonstrate the first sequence (described in the example above) by writing out the steps on

the board and having students execute the moves in order. Repeat this activity with students selecting new movements and their sequence.

Part two: Adding Loops (repeating steps)
In coding, we use loops as an instruction to repeat steps. This makes the code more efficient and easier to read than if we wrote out our steps again and again (compare the difference between saying "take one step forward three times" and "take one step forward then take one step forward then take one step forward").

For example, we can make the sequence from our first exercise more efficient by adding a loop. So, instead of writing out the step "clap your hands" three times, we only need to write it out once and use a loop to communicate that we are going to repeat that step three times.

We will represent our loop with a circle drawn on the board. Every step contained within the circle will be repeated. We will indicate the number of times that we are repeating the steps by writing a number next to the loop circle. It would look something like this:

Have students execute the moves in these instructions. The result should be the same as our first sequence, but with fewer steps. Repeat this activity with students selecting new movements and their sequence. Have them experiment with how many steps they place inside a loop and the number of repetitions that they assign to the loop.

What happens if they place a loop around the entire set of instructions? (Answer: the entire set of instruction gets repeated.)

What happens if you nest loops (e.g., put loops inside of loops)? If we place a loop around the example above:

Now students would have to repeat the sequence (stand up, clap hands three times, sit down) twice (stand up, clap hands three times, sit down, stand up, clap hands three times, sit down). Note that this loop treats the entire sequence as a unit (e.g., the new loop does not mean that you clap your hands six times in a row before moving to the next step).

Part three: Concurrent Steps

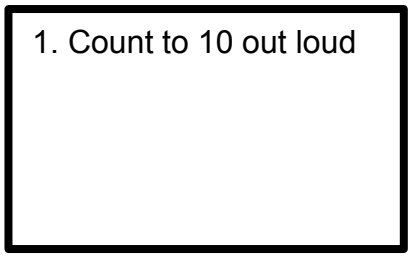
In our previous two exercises, we are executing our steps one at a time, in order. But what do you do if you want to tell your program to do two things at the same time?

We will indicate concurrent programs by drawing the steps that we want to have happen at the same time in side-by-side boxes.

For example, say that we want to instruct our program to count to ten out loud at the same time that they run our original stand up - clap your hands – sit down program:

Event 1:

Event 2:



Note: Students may suggest describing the program for Event 2 as a series of steps (e.g., 1. Say “one” out loud, 2. Say “two” out loud, etc.) and this is also acceptable.

Demonstrate the above example to by writing out the concurrent sequences on the board and having students execute the instructions from both Event 1 and Event 2 at the same time.

Repeat this activity with students selecting the movements and their sequences.

Part four: Putting it all together

Now that students have practiced using sequential and concurrent events, loops and nested loops, encourage them to apply this learning by building a dance (consider: how would you describe the instructions for dances like the hokey pokey?). This can be done with breakout groups who then present their dance codes to the class for everyone to try by following their code.

If a student group's dance instructions are unclear or result in a dance different than what the students had intended, then there is a great opportunity as a class to "debug" the dance moves. Usually this involves simplifying instructions (e.g., making sure that each step only describes one action) and separating complicated instructions into shorter, concurrent events.

For example: if a dance includes a step such as "raise your right arm and shake it" the debugging process might look like this:

- Clarify: are you raising your right arm **AND THEN** shaking it? If so, this step should be separated into two steps in the same sequence (1. Raise your right arm; 2. Shake your right arm)
- Are you raising your right arm **WHILE** shaking it? If so, then these steps should be broken into concurrent events

Event 1:	**Event 2:**
1. Lift right arm	1. Shake right arm

Remind students that when they are planning their code, that they can write out their algorithms using this sort of format to help map out their instructions on paper before moving to the coding software.

The **Grow with Code Handout** provides additional exercises for students to describe instructions using sequential, concurrent, and repeating events.

<u>Coding Math Activity</u>

The goal of the coding activity is to use the block-code application Scratch to create a plant sprite that will "grow" (increase in size) when it touches factors (sunlight, water) that support plant growth. This code will allow students to practice using loops and nested events.

A detailed step-by-step guide to building this program in Scratch, including opportunities for

extension, is described in the **Grow with Code Coding Guide.**

For students who are pre-literate or who are absolute beginners to coding, refer to the simplified coding guide — **Grow With Code Variation — Scratch Jr. Coding Guide for Beginners** — In this simplified coding guide, we will use Scratch Jr. to create a scene with a background, a sun that moves when you click it, a raincloud that moves when you click it, a plant that grows larger when you click it, and flowers that appear.

The simplified coding lesson uses Scratch Jr. instead of Scratch. Scratch Jr. uses pictograms on their coding blocks instead of text to illustrate that block's functions. This program will be useful for students whose literacy skills are not yet at a level where they can easily understand the language written on Scratch blocks, or as practice to familiarize students with the logic of block coding and building basic algorithms.

Closure and Assessment

By the end of this lesson, students should be able to identify coding elements as sequential, concurrent, or nested and describe how these different instruction arrangements affect how instructions (algorithms) are executed. Students should be able to identify loops and describe how the instruction contained within loops are repeated.

For assessment, collect the **Grow with Code Handout** from the students. Review their work to ensure that they understood the concepts of sequential events, concurrent events, and loops by evaluating their responses using the answer key.

Adaptations

- The actions used in the unplugged activity can be adapted to fit students' mobility and/or accessibility needs
- The symbol system used to indicate event algorithms and loops can be adapted to fit students' accessibility needs (including arranging physical images of steps)
- A variant coding guide is included among resources for this lesson for use by pre-literate students who may find the text on Scratch coding blocks inaccessible.

Extensions

- Students who finish their codes early can add more factors that affect their plant sprites. Details for opportunities for extension are included in the **Grow with Code Coding Guide**.