SCIENCE NORTH SCIENCE NORD
SUDBURY, ONTARIO, CANADA

Cell Explorers	Grade 8

Lesson Plan

Coding Tool	Scratch
Time Required	2-3 periods

Math Curriculum Connections

Algebra

Overall Expectations

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 code, including code that involves the analysis of data in order to inform and communicate decisions

C3.2 read and alter existing code involving the analysis of data in order to inform and communicate decisions, and describe how changes to the code affect the outcomes and the efficiency of the code

Science Curriculum Connections

Grade 8
Cells

- Cells are the basis of life.

Overall Expectations

3. Demonstrate an understanding of the basic structure and function of plant and animal cells and cell processes

Specific Expectations

3.1 Identify structures and organelles in cells, including the nucleus, cell membrane, cell wall, chloroplasts, vacuole, mitochondria, and cytoplasm, and explain the basic functions of each.

.3 Compare the structure and function of plant and animal cells.

Description

Through a series of hands-on and coding activities, students will investigate cell organelles and their functions. Students will learn that many cellular processes are governed by rules or patterns that can be interpreted and expressed in code as conditional statements and loops. They will illustrate this learning by creating a playable Scratch game based on these patterns in cell functions.

Success Criteria

- 8th grade students will be able to identify the organelles found in animal cells and describe their basic function
- 8th grade students will be able to describe algorithms using pseudocode and block coding, including loops and conditional statements. They will be able to demonstrate how code can be used to execute decisions using conditional statements.

Materials and Media

- Computer or Tablet with access to Scratch (either browser version or app)
- A large number of similarly sized sortable objects that can be easily sorted into 3 groups (e.g., coloured balls or pompoms)
- Nucleobase Matching Coding Guide
- Plant or Animal Coding Guide
- Cell Algorithm Handout

Computational Thinking Skills

This lesson approaches code in terms of decomposition, or breaking complex problems up into smaller parts to make it easier to tackle. There is also special focus placed on conditional statements. Conditional statements are used in coding to execute a condition if a statement is true (this can be considered similar to helping a computer to make a programmed decision based on given factors or information).

In the unplugged activity, students are encouraged to consider organelle functions in terms of conditional (if/then) statements and practice writing out the "rules" that govern these functions as pseudocode. This is a helpful step for understanding how coding languages, especially nesting elements common to loops and conditional statements, fit together, and a great tool to use when students are planning code.

In the online coding activity, students will use Scratch, to build and modify a game that uses variables, loops, and conditional statements to transform inputs (the game player's key presses) into score increases for correct choices, and score decreases for incorrect choices. The game topic reinforces students' learning about cells (with alternate versions described to suit your classroom's focus).

The **Nucleobase Matching** and **Plant or Animal** coding guide documents for this lesson include a detailed step-by-step procedure for using Scratch.

Introduction

Introduction to Cells and Organelles

Cells are the basic unit of life. They contain smaller parts, called organelles, that are specialized to perform specific tasks.

Start by reviewing the organelles found in animal cells and their basic functions.

- Cell membrane: controlling what enters and exits the cell
- Cytoplasm: not *technically* an organelle, but itcontains/suspends organelles inside the cell
- Ribosomes
- Endoplasmic Reticulum: comes in smooth (SER) and (RER) forms. SER produces lipids (fats) that form the cell membrane; RER supports protein production (for proteins to leave the cell)
- Mitochondrion: energy production
- Ribosome: build proteins by decoding information (from RNA) and placing amino acids (the building blocks of proteins in the right order according to the decoded information.
- Golgi Apparatus: protein modification and packaging
- Lysosome: protein destruction
- Vacuole: storage for waste, water, or nutrients (depending on the type of cell)
- Nucleus: DNA Storage

This lesson will look at how we can describe organelle functions based on a set of rules, or an algorithm. We will take a closer look at the cell membrane, ribosomes, and the nucleus.

Introduction to Conditional Statements

We use conditional statements every day without thinking about it. A conditional statement can be thought of as a decision based on the answer to a question. Think of the decision to wear your mitts.

The question that you're answering might be:

"Is it cold outside?" or "are my hands cold?" or "will my hands be cold if I go outside?"

Let's say the question we're answering to decide whether to wear mitts is "are my hands cold?".

This question can be answered with a yes (my hands are cold) or no (my hands are not cold). In real life, you will probably decide to put your mitts on if your hands are cold and to not put your mitts on if your hands are not cold.

In code, this decision can be represented as a conditional statement, known as an if/then statement:

If my hands are cold, **then** I put my mitts on.

In this case the **if** part of the statement is the condition being met (hands are cold) and **then** is the program that is executed as a result of the condition being met. We can even code for the decision to leave your mitts off because your hands aren't cold with an **else** statement. It might be helpful to think of **Else** as "otherwise".

If my hands are cold, **then** I put my mitts on. **Else,** I don't put my mitts on.

We can also have more than one condition that can lead to a decision. For example, we came up with a few questions that might lead us to put mitts on. We could add a rule to describe that we would put mitts on if it's cold out, even if our hands aren't cold (yet) using an **else-if** statement. It might be helpful to think of **else-if** as "OR if".

If my hands are cold, **else if** it's cold out, **then** I put my mitts on. **Else,** I don't put my mitts on.

The way that we're describing these conditions in plain language is a form of what's known as pseudocode. Pseudocode is a useful tool for planning out a coding program before jumping into a coding language.

We're going to use pseudocode to look at the decisions that different organelles use to perform their tasks inside animal cells.

Action

Unplugged Activity

The cell membrane protects the cell and controls molecules that enter the cell (like nutrients) and exit the cell (like waste). Some of these molecules pass easily through openings in the membrane (passive transport, such as diffusion or osmosis). Other molecules need to be paired with a carrier molecule to get help to move across the membrane (active transport).

Use part 1 of the **Cell Algorithms Handout** to outline the rules for the cell membrane activity as conditional expressions. This will be written as pseudocode, for example:

The goal of our cell membrane program is to check if a ball is blue or red or yellow*. It will let individual red balls pass, but not individual blue or yellow balls. Blue and yellow balls can only pass if they are paired (one blue ball + one yellow ball together).

So, as pseudocode:

if ball is red
    then pass through membrane

else if ball is (paired) blue AND yellow
    then pass through membrane

else
    then do not pass through membrane

*The objects that your class is sorting and the criteria from sorting can be whatever you like.

Now, try executing the rules to move the balls from one side of the class to the other. Divide students into three groups:

1) Group 1 (outside of the cell) will collect balls (one ball per student at a time) to bring to the cell membrane.

2) Group 2 will act as the cell membrane. They will immediately pass red balls through to Group 3 students (inside the cell), and will hold blue or yellow balls until they have a pair (1 yellow ball + 1 blue ball). Once they have a pair, they will pass both balls to group 3 students.
3) Group 3 (inside the cell) will retrieve balls from cell membrane and place them into a bin (max 1 red ball or 1 yellow-blue pair at a time).

Opportunity for extension: we can add additional rules to the game to make it more challenging. For example: students who are acting as the cell membrane can only hold a maximum of two balls at a time. How would you express this as a conditional statement in pseudocode?

Example:

Pseudocode	**Explanation**
repeat until BallValue = 2 **receive** one ball; **for** every ball received increase [BallValue] + 1;	This is a **loop** with a counter. Every time a student receives a ball, the BallValue variable increases by +1. Once the BallValue variable equals 2, the code will exit the loop and move on to the next section of code.
check BallColours **if** BallColours do not match pass balls through membrane; **else** do not pass balls;	This is a **conditional statement**. The student checks the two balls that they are holding to make sure that the pair is non-matching (e.g., Yellow and Blue). If the colours do not match, the **if** statement's condition is TRUE and the balls can pass the membrane according to the rules of the program. If the colours do match, the **if** statement's condition is FALSE and so the program will execute instead the output for the **else** condition, which is to not allow the balls to pass the membrane.

Complete the Cell Algorithms Handout for more practice breaking down the steps of cell functions as pseudocode. The second example on the handout looks specifically at ribosomes and how they build chains of amino acids according to three-letter groupings known as

codons. Each letter in a codon represents a nucleobase, which is the "language" with which we encode and store information in our DNA. There are four bases in DNA: A (adenine), C (cytosine), G (guanine), T (thymine) — we'll be referring to these bases by their letters this lesson.

Ribosomes read a transcribed form of DNA called mRNA to build proteins from building blocks called amino acids. mRNA also uses four bases, just like DNA. The only difference is that T (thymine) is replaced by a new base U (Uracil). So, the four bases of mRNA are: A, C, G, and U.

These bases can make 64 different combinations of groupings of three bases to code for 20 types of amino acids. To keep things simpler, our handout only shows us four of these combinations (called **codons**), along with their associated amino acid (and a shape):

UUA = Phe (triangle)
AUG = Met (circle)
CGC = Arg (star)
UCU = Ser (square)

Translate the following string of 10 codons, what would be the resulting sequence of amino acids? What would be the resulting sequence of shapes?

AUGCGCCGCUCUUUAUCUUUACGCUUACGC

Answer:

Met-Arg-Arg-Ser-Phe-Ser-Phe-Arg-Phe-Arg

OR

Circle – Star – Star – Square – Triangle – Square – Triangle – Star – Triangle – Star

What set of rules did you use to translate the codons into their amino acids? How would you write this as pseudocode?

Example:

Repeat until all nucleobases in sequence are coded:
    Read nucleobases in groups of three starting from the first base on the left;
        if grouping is AUG
            then place Met (or circle);

else if grouping is CGC
then place Arg (or star);

else if grouping is UCU
then place Ser (or square);

else
place Phe (or triangle);

Coding Activity

Recall that DNA is stored in the nucleus of the cell and that DNA is "written" in four letters, or bases: A, T, G, and C. If you picture DNA molecules as a ladder, each rung on the ladder is actually two of these bases paired together. A is always paired with T. G is always paired with C.

We are going to use **Scratch** to create a game that uses the same matching rules for DNA bases. (See note below for an alternate version of this game that can be built).

Here is what we want our game to do:

- Each letter A, T, G, and C will be assigned to an arrow key on a keyboard (or touch button on iPad);
- The game will randomly display one letter (A, T, G, or C) at a time on the screen;
- The player will press the arrow key for the letter that **pairs** with the letter on the screen (so, if the screen is showing A, then the player should press the arrow assigned to T);
- If the player guesses correctly, increase score by one point
- If the player is incorrect, decrease score by one point.

A detailed step-by-step guide to building this game is described in the **Nucleobase Matching Coding Guide.** .
An example version of this code can be viewed and played here:
https://scratch.mit.edu/projects/438699096/editor/

If we think in terms of pseudocode like we did with earlier activities, the game's algorithm breaks down as follows:

Pseudocode	Explanation
Repeat forever	This is a **loop** that will contain the program. We want our game to be

	checking and responding to keys pressed by the player indefinitely.
if letter A is on-screen **then**	This is a **conditional** expression. Because we have four programs running concurrently and checking on loop to see which letter costume is being displayed, we need a condition to trigger this program. When A is displayed (and not T or C or G) then this condition is met and the program for A will be executed (and not the program for T or C or G).
if right arrow pressed **then** change score by +1	This is a conditional statement (nested in the previous one!). If the right arrow key (the key assigned to T, or the base that pairs with A) is pressed by the player, then the correct key has been pressed and the score counter will increase the score value by 1.
if up arrow, down arrow, or left arrow pressed **then** change score by -1	This is another conditional statement (nested at the same level as the previous conditional). The right arrow key is the only correct key, so if the player presses any other key, an incorrect key has been pressed and the score counter will decrease the score value by 1.

Alternate version: Plant or Animal Cell?

This game can be easily rewritten to be a study tool for learning the organelles and identifying which can be found in plant cells, animal cells, or both.

Here is what we want our game to do:

- Three options (Plant Cell, Animal Cell, Both) will be assigned to arrow keys on the keyboard (or touch buttons on iPad).
- The game will randomly display one name of an organelle at a time on the screen.
- The player will press the arrow key for the type of cell that corresponds with that organelle (Plant, Animal, or Both)
- If the player guesses correctly, increase score by one point.

- If the player is incorrect, decrease score by one point.

This version is otherwise built in the same fashion as the Nucleobase Matching Game. It codes for fewer playable keys (3 instead of 4) but has more code overall (one program for each organelle that you include in the game). Sample code is included in the **Plant or Animal Coding Guide.**

Closure and Assessment

By the end of this lesson, students should be able to recognize animal cell organelles and describe their basic functions. Students should be able to modify conditional statements to change the output of a digitally created program, playtest their code, and iterate.

For assessment, collect the "Cell Algorithms" handouts from the students. Review their work to ensure that they understood the concepts of algorithms and conditional statements by appropriately describing the steps required for cell membranes and ribosomes to perform their tasks.

Adaptations

- Offline cell membrane sorting activity can be done seated as a class or seated individually at their desks
- The objects used for the cell membrane sorting activity can be done using objects sorted by characteristics other than colour (such as shape or type)

Extensions

- Students who finish early can make modifications to their code to see how it affects gameplay. Modifications can include adjusting the challenge of the game by changing the value in the wait block or the score values for different conditions or adjusting the aesthetics of the game by changing the sounds played or the look of the game.

Additional Resources

- Scratch.mit.edu – Scratch is a free resource and no account is required to create a program; however, an account is required to save your work.