Algorithms and Programming Module 5
Lesson 5.1-5.5

Context of the Lesson	
The Big Idea: Analyzing, correct, and improve (debug) an algorithm	
Prerequisite Knowledge and Skills: • Knowledge of sequencing, loops, events, conditionals, and variables • Basic understanding of computing language	Connections to SOLs: • Computer Science 5.1 • Computer Science 5.2 • Computer Science 5.3 • Computer Science 5.4 • Computer Science 5.5 • Math 5.2 • Math 5.3 • Math 5.4 • Math 5.5 • Math 5.7 • Math 5.18 • Math 5.19 • English 5.7 c, de, e

Objectives of the Lesson	Formative Assessment
Learning Targets (I can...): • I can construct sets of step-by-step instructions (algorithms) both independently and collaboratively, using sequencing; using loops; using variables to store and process data; performing number calculations on variables; and using conditionals. • I can construct programs to accomplish a task as a means of creative expression using a block or text based programming language, both independently and collaboratively using sequencing; using loops; using variables; using mathematical operations; and using conditionals. • I can analyze, correct, and improve (debug) an algorithm that includes sequencing, events, loops, conditionals, and variables. • I can analyze, correct, and improve (debug) an algorithm that includes sequencing, events, loops, conditionals, and variables. • I can create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map). • I can break down (decompose) a larger problem into smaller subproblems, both independently and collaboratively.	• Extended time on assignment • Verbal testing • Selective student seating • Group and individual activities • Instructional aids • Written testing

Materials	
• Visual Aid for instructor to write on (e.g., chalkboard, projector, dry erase board, easel) • Handouts and writing utensil for students (optional) • Computer (optional) • Means to play video (optional)	

Lesson Structure and Activities

Warm Up *[5-10min]* , answers to be written out by instructor on visual aid

Ask: What are some routines you follow on a daily basis?

- Responses will very. (morning routine before school, clean-up time, and dismissal)

Launch (Engage) *[5-10min]* :Teacher Directed Instruction:

An **algorithm** is a sequence of steps designed to accomplish a specific task. Algorithms are translated into programs, or code, to provide instructions for computing devices. Algorithms are used to accomplish a certain task. Examples of algorithms in everyday life include cooking recipes, directions to a location. Instructions on how to assemble an object. Algorithms in Computer Science are used to do various things, like finding the shortest route to a location or recommendation for what movie you should watch next. Algorithms and programming control all computing systems, empowering people to communicate with the world in new ways and solve compelling problems. The development process to create meaningful and efficient programs involves choosing which information to use and how to process and store it, breaking apart large problems into smaller ones, recombining existing solutions, and analyzing different solutions.

Algorithms are designed to be carried out by both humans and computers. People follow and create processes as part of daily life. Many of these processes can be expressed as algorithms that computers can follow. Other examples of algorithms include making simple foods, navigating a classroom, and daily routines like brushing teeth. Just as people use algorithms to complete daily routines, they can program computers to use algorithms to complete different tasks. Algorithms are commonly implemented using a precise language that computers can interpret.

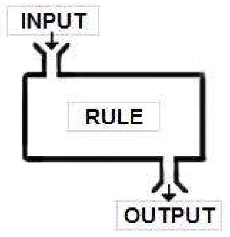
Different algorithms can achieve the same result. Some algorithms are more appropriate for a specific context than others. Different algorithms can be used to tie shoes or decide which path to take on the way home from school. While the end results may be similar, they may not be the same: in the example of going home, some paths could be faster, slower, or more direct, depending on varying factors, such as available time or the presence of obstacles (for example, a barking dog). Algorithms can be expressed in noncomputer languages, including natural language, flowcharts, and pseudocode.

Ask: What are some other examples of algorithms you follow on a daily basis that can be achieve differently but have the same result?

- Responses will very. (walking home from school vs. driving in a car; mailing a letter vs. email or text; walking somewhere vs running there)

Algorithms can also be used in MATH.

An example of a math algorithm:***<u>Blackboard Teaching Exercise</u>****
On blackboard draw:



What is the RULE/FUNCTION/ALGORITHM that takes these inputs to output these outputs?

INPUT	OUTPUT
10	6
15	11
5	1

Answer: $x - 4$

INPUT	OUTPUT
10	5
0	0
30	15

Answer: $x/2$

Using handout 5.1 have students do question 1 by themselves.

Computer programs store and manipulate data using **variables**. Variables store values to be used later. This is just like variables in algebra except they can store other types of values, like a list of values. You can add, subtract, multiply, and divide values in variables using the operators +,-,*, and /.

Information in the real world can be represented in computer programs. Programs store and manipulate data, such as numbers, words, colors, and images. The type of data determines the actions and attributes associated with it. Different actions are available for different kinds of information. For example, sprites (character images) can be moved and turned, numbers can be added or subtracted, and pictures can be recolored or cropped.

Programming languages provide variables, which are used to store and modify data. The data type determines the values and operations that can be performed on that data. Variables are the vehicle through which computer programs store different types of data. Data types vary by programming language, but many have types for numbers and text. Examples of operations associated with those types are multiplying numbers and combining text. Some visual, blocks-based languages do not have explicitly declared types but still have certain operations that apply only to particular types of data in a program.

An example of a math variable. ***Blackboard Teaching Exercise****

Emma has a bakery and sells cupcakes and cookies. She made $22 for selling 4 cupcakes and $6 worth of cookies. Write a number sentence that describes how she made the money using x to represent how much money she made from selling the cupcakes.

Answer: $4x+6=22$

Using handout 5.1 have students do question 2 by themselves.

An example of a math variable. ***Blackboard Teaching Exercise****

If $x = 5$, $y = 3$, and $z = x*y$, then what is the real value of z?
Answer: 15

Using handout 5.1 have students do question 3 by themselves.

Control structures specify the order in which instructions are executed within an algorithm or program.

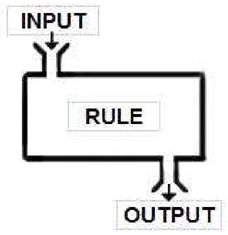
Computers follow precise sequences of instructions that automate tasks. Program execution can also be nonsequential by repeating patterns of instructions and using events to initiate instructions. Computers follow instructions literally. Examples of sequences of instructions include steps for drawing a shape or moving a character across the screen. An event, such as the press of a button, can trigger an action. Simple loops can be used to repeat instructions.

Control structures, including *loops, event handlers, and conditionals*, are used to specify the flow of execution. Conditionals selectively execute or skip instructions under different conditions. Different types of loops are used to repeat instructions in multiple ways depending on the situation. Examples of events include mouse clicks, typing on the keyboard, and collisions between objects. Event handlers are sets of commands that are tied to specific events. Conditionals represent decisions and are composed of a Boolean condition that specifies actions based on whether the condition evaluates to true or false. Boolean logic and operators (e.g., AND, OR, NOT) can be used to specify the appropriate groups of instructions to execute under various conditions.

A **for loop** allows you to repeat some code a set number of times. This allows you repeat a set of instructions for an algorithm that requires a set of instruction to be repeated z set amount of times. For example, if you want to find the values of y for the values of x 1 to 5, you first start by calculating the value of y where x is 1, then you do the same for the other numbers up to 5.

Conditionals, also called if statements, are used to do certain steps only if a certain condition is met. The most common way people encounter them is in English sentences. For example, "If it is raining outside, then you bring an umbrella". It is common sense that the part after the word then is only done if the statement between if and then is true.

An example of a math conditional. ***Blackboard Teaching Exercise****



Write on blackboard this rule: if $x > 10$, then $x + 1$

X INPUT	OUTPUT
17	?
12	?
9	?
10	?

Answer:

X INPUT	OUTPUT
17	18 because it is (17+1)
12	13 because it is (12 +1)
9	9 because 9 is less than x>10
10	10 because 10 is less than x>10

Using handout 5.1 have students do question 4 by themselves.

Modularity involves breaking down tasks into simpler tasks and combining simple tasks to create something more complex.

Complex tasks can be broken down into simpler instructions, some of which can be broken down even further. Likewise, instructions can be combined to accomplish complex tasks. Decomposition is the act of breaking down tasks into simpler tasks. An example of decomposition is preparing for a party: it involves inviting guests, making food, and setting the table. These tasks can be broken down further. For example, setting the table involves laying a tablecloth, folding napkins, and placing utensils and plates on the table. Another example is breaking down the steps to draw a polygon. Composition, on the other hand, is the combination of smaller tasks into more complex tasks. To build a city, people build several houses, a school, a store, etc. To create a group art project, people can paint or draw their favorite ocean animal, then combine them to create an ecosystem.

Programs can be broken down into smaller parts to facilitate their design, implementation, and review. Programs can also be created by incorporating smaller portions of programs that have already been created. Decomposition facilitates aspects of program development, such as testing, by allowing people to focus on one piece at a time. Decomposition also enables different people to work on different parts at the same time. An example of decomposition at this level is creating an animation by separating a story into different scenes. For each scene, a background needs to be selected, characters placed, and actions programmed. The instructions required to program each scene may be similar to instructions in other programs.

[Source: https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-CS-Framework-Statements-Concepts-View.pdf]

Explore *[25min]* :Joint/Guided Practice | Student Practice:

Group activity building off the lecture discussed above. Different activities to choose from with different topics and difficulty levels.

5.2 Scratch Exercise

Summarize *[15min]* :Debrief :

Small Group or individual exercise. Ask these questions, give time for answer, entire group discussion of answers.

Extensions:	