Lindenwold
Public Schools

Unit 3: Intro to Programming
December-January

Targeted Standards: K12 Computer Science Standards

- Computing Systems - Devices, Hardware and Software, Troubleshooting
- Data and Analysis - Collection, Visualization and Transformation, Inference and Models
- Algorithms and Programming - Algorithms, Variables, Control, Modularity, Program Development
- Impacts of Computing - Culture, Social Interactions, Safety, Law, and Ethics

(also reference CSTA K-12 Computer Science Standards)

Rationale and Transfer Goals : This unit introduces students to programming in the JavaScript language and creating small applications (apps) that live on the web. This introduction places a heavy emphasis on understanding general principles of computer programming and revealing those things that are universally applicable to any programming language.

Enduring Understandings:

- Creative development can be an essential process for creating computational artifacts.
- Computing enables people to use creative development processes to create computational artifacts for creative expression or to solve a problem.
- Multiple levels of abstraction are used to write programs or create other computational artifacts
- Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- People write programs to execute algorithms.
- Programming is facilitated by appropriate abstractions.

Essential Questions:

- Why do we need algorithms?
- How is designing an algorithm to solve a problem different from other kinds of problem solving?
- How do you design a solution for a problem so that is programmable?
- What does it mean to be a "creative" programmer?
- How do programmers collaborate?

Content/Objectives		**Instructional Actions**	
Content ***What students will know***	**Skills** ***What students will be able to do***	**Activities/Strategies** ***How we teach content and skills***	**Evidence (Assessments)** ***How we know students have learned***
● Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages. ● People write programs to execute algorithms. ● Algorithms can solve many but not all computational problems. ● Multiple levels of abstraction are used to write programs or create other computational artifacts ● Programs can be developed for creative expression, to satisfy	● Assess the clarity of a set of instructions expressed in human language. ● Create a set of instructions in human language for building a simple LEGO block arrangement. ● Identify connections between the ability to program and the ability to solve problems. ● Describe the ambiguities inherent in human language and the ways programming languages seek to remove those ambiguities.	● Concept Innovation ● Unplugged ● Algorithms ● Turtle Programming	● Consider the algorithm you designed for today's activity. Identify two instances where there may be multiple ways to interpret your instructions and suggest improvements that could be made to improve their clarity. ● Describe the features of a programming language that make it different from the language you are used to using in everyday life. Explain why a programming language must be created in this way.

personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society). ● Programming is facilitated by appropriate abstractions.	● Trace programs written in the "Human Machine Language" ● Develop an algorithm to find the smallest playing card in a row of cards ● Express an algorithm in the "Human Machine Language" ● Identify the properties of sequencing, selection and iteration the "Human Machine Language" ● Evaluate the correctness of algorithms expressed in the "Human Machine Language" ● Develop an algorithm to solve a new problem with playing cards ● Express an algorithm in the Human Machine Language ● Identify Sequencing, Selection and Iteration in a program written the Human Machine Language ● Describe the properties of the Human Machine Language that make it a "low level" language.		● Write a human machine language program that: Repeatedly shifts the left hand to the right until it finds a 5 or 6 The program should stop when the left hand is at (or past) the end of the list, or it finds a 5, or it finds a 6. ● This lesson introduced the notion of "efficiency" in programming, and that it might mean different things at different times. Think of an example outside of computer science in which you have heard the term "efficiency" and compare it to the ways we talked about efficiency in programming. In what ways is the meaning of "efficiency" the same? In what ways is it different? ● Today we solved a series of problems with a limited set of commands (only 4). Give at least one reason why it's useful to learn how to solve, and

	● Solve simple programming challenges when the set of allowed commands is constrained. ● Explain considerations that go into “efficiency” of a program. ● Use App Lab to write programs that create simple drawings with “turtle graphics.” ● Work with a partner to program a turtle task that requires about 50 lines of code. ● Justify or explain choices made when programming a solution to a turtle task. ● Recognize functions in programs as a form of abstraction. ● Write a program that solves a turtle drawing problem using multiple levels of abstraction (i.e. functions that call other functions within your code). ● Explain why and how functions can make code easier to read and maintain.		program solutions to problems with a limited set of commands. ● In your own words explain at least one reason why programming languages have functions. ● In the Create Performance Task you will be asked to identify an abstraction in your program and explain how it helps manage the complexity of the program. Functions are a form of abstraction. Pick a function you wrote in your solution to the 3x3 square problem and explain how it helps manage the complexity of your program. ● It is said that functions with parameters generalize the behavior of a more specific command. Explain what this sentence means to you using the difference between turnLeft() and turnLeft(angle).

	● Define and call simple functions that solve turtle drawing tasks. ● Write a complete program with functions that solve sub-tasks of a larger programming task. ● Explain how functions are an example of abstraction. ● Use a “top-down” problem-solving approach to identify sub-tasks of a larger programming task. ● Use parameters to provide different values as input to procedures when they are called in a program. ● Use API documentation to assist in writing programs. ● Define an API as the set of commands made available by a programming language. ● Write functions with parameters to generalize a solution instead of duplicating code.		● “Abstraction” is often used to indicate cases where we focus on a general case and ignore a specific instance of a problem. Given this meaning of the word, how are functions with parameters an example of abstraction? ● When breaking a problem down, you often encounter elements that you want to use repeatedly in your code. Sometimes it's appropriate to write a new function; at other times it's appropriate to write a loop. There is no hard-and-fast rule as to which is better, but what do you think? What kinds of circumstances would lead you to writing a function versus using a loop?

	● Identify appropriate situations for creating a function with parameters. ● Use random numbers as inputs to function calls for the purpose of testing. ● Add parameters to a function in an existing piece of code to generalize its behavior. ● Use a loop in a program to simplify the expression of repeated tasks. ● Identify appropriate situations in a program for using a loop. ● Use random values within a loop to repeat code that behaves differently each time it is executed. ● Write programs that address one component of a larger programming problem and integrate with other similarly designed programs. ● Collaborate to break down a complex programming problem into its component parts.		

	● Use code written by other programmers to complete a larger programming task.		

<u>Spiraling for Mastery</u>

Where does this unit spiral back to other units from this or previous years in order to ensure that students retain mastery of what they've learned?

Content or Skill for this Unit	Spiral Focus from Previous Unit	Instructional Activity
● People evaluate and select algorithms based on performance, reusability, and ease of implementation. Knowledge of common algorithms improves how people develop software, secure data, and store information. ● Data structures are used to manage program complexity. Programmers choose data structures based on functionality, storage, and performance tradeoffs.	● Algorithms affect how people interact with computers and the way computers respond. People design algorithms that are generalizable to many situations. Algorithms that are readable are easier to follow, test, and debug. ● Programmers create variables to store data values of selected types. A meaningful identifier is assigned to each variable to access and perform operations on the value by name. Variables enable the flexibility to represent different situations, process different sets of data, and produce varying outputs. ● Applications store data as a representation.	● The Need for Programming Languages ● Creativity in Algorithms

● Data can be composed of multiple data elements that relate to one another. For example, population data may contain information about age, gender, and height. People make choices about how data elements are organized and where data is stored. These choices affect cost, speed, reliability, accessibility, privacy, and integrity.	Representations occur at multiple levels, from the arrangement of information into organized formats (such as tables in software) to the physical storage of bits. The software tools used to access information translate the low-level representation of bits into a form understandable by people.	● APIs and Using Functions with Parameters

21st Century Skills: What are the 21st Century Skills that are a part of this unit, and where are they experienced?

- Global awareness
- Creativity and Innovation
- Critical Thinking and Problem Solving
- Communication and Collaboration
- Information Literacy
- Flexibility and Adaptability
- Initiative and Self Direction

These skills are experienced throughout unplugged and plugged activities that will involve individual, group, and whole class discussion.

Key resources: What are the resources that are essential for this unit (may also be listed in "Activities/Strategies")?

- Unit 3 - Code.org Computer Science Principles Curriculum
- Turtle Programming
- Under the Sea Challenge