Learning Target: I can investigate why programming languages exist

Success Criteria/Objectives:

- ☐ Present an overview of assigned programming language
- ☐ Explain the qualities that differentiate natural languages and programming languages
- ☐ Justify the existence of programming languages to precisely communicate instructions
- ☐ identify and explain the two types of programing language
- ☐ explain the two categories of low-level language
- ☐ define 'compiler' and 'interpreter' and explain what each is
- ☐ provide examples of interpreter and compiler language

Standards:

4-6.CT.6 Compare two or more algorithms and discuss the advantages and disadvantages of each for a specific task. 7-8.CT.6 Design, compare and refine algorithms for a specific task or within a program. 9-12.CT.6 Demonstrate how at least two classic algorithms work, and analyze the trade-offs related to two or more algorithms for completing the same task.

Key Ideas:

A program is a set of instructions that tells a computer what to do in order to come up with a solution to a particular problem. Programs are written using a programming language. A programming language is a formal language designed to communicate instructions to a computer. There are two major types of programming languages: low-level languages and high-level languages. Low-level languages are referred to as 'low' because they are very close to how different hardware elements of a computer actually communicate with each other. Low-level languages are machine oriented and require extensive knowledge of computer hardware and its configuration. There are two categories of low-level languages: machine language and assembly language.

1. **Introduction**

   What is a language?

   How do computers know what to do?

2. **Mini Lesson:**

If you're listening to morning announcements at school and hear that there will be a practice fire drill, how does this information process into your knowledge of understanding? Draw how this information flows. For example, draw a circle with the words 'Fire drill announcement' inside, then an arrow, then a circle with the words 'Exit the building at the sound of the alarm.' Think of a more complicated command structure: learning to cook a dish, or doing a drill in sports. Have groups first write this structure, then draw on a flow-chart map on chart paper, then share with the class.

3. **Activity**

a. Paired Reading - Programming Languages (see reading below)

b. Presentation on a programming language - student pairs are assigned to one programming language and give an overview of its history use and example code (C, Python, C++, Java, SCALA, C#, R, Ruby, Go, Swift, JavaScript)

4. **Summary**

What are three things you learned, two things you're still curious about, and one thing you don't understand?

5. **Out of class practice/homework**

Readings - coding languages computer programming language

6. **Resources**

a. https://study.com/academy/lesson/machine-code-high-level-language-lesson-plan.html
b. https://www.w3schools.com/ai/ai_history_languages.asp
c. https://ny.pbslearningmedia.org/resource/first-programming-language-crash-course-cs/first-programming-language-crash-course-cs/
d. Coded bias teacher guide
e. https://chortle.ccsu.edu/Java5/index.html#03
f. https://studio.code.org/s/csp3-2021/lessons/5
g. http://www.nysed.gov/common/nysed/files/programs/curriculum-instruction/computer-science-digital-fluency-standards-k-12.pdf
h. https://www.windham-schools.org/docs/DOK%20Wheel%20Slide%20for%20Teachers-0.pdf

Reading: **Machine Code and High-level Languages: Using Interpreters and Compilers**

VOCABULARY
Programming language
Low-level language
Machine language
Machine code
Assembly language
High-level language
Compiler
Interpreter

Machine language, or machine code, is the only language that is directly understood by the computer, and it does not need to be translated. All instructions use binary notation and are written as a string of 1s and 0s. A program instruction in machine language may look something like this:

10010101100101001111101010011011100101
Technically speaking, this is the only language computer hardware understands. However, binary notation is very difficult for humans to understand. This is where assembly languages come in. An assembly language is the first step to improve programming structure and make machine language more readable by humans. An assembly language consists of a set of symbols and letters. A translator is required to translate the assembly language to machine language. This translator program is called the 'assembler.' It can be called the second generation language since it no longer uses 1s and 0s to write instructions, but terms like MOVE, ADD, SUB and END.

Many of the earliest computer programs were written in assembly languages. Most programmers today don't use assembly languages very often, but they are still used for applications like operating systems of electronic devices and technical applications, which use very precise timing or optimization of computer resources. While easier than machine code, assembly languages are still pretty difficult to understand. This is why high-level languages have been developed.

A high-level language is a programming language that uses English and mathematical symbols, like +, -, % and many others, in its instructions. When using the term 'programming languages,' most people are actually referring to high-level languages. High-level languages are the languages most often used by programmers to write programs. Examples of high-level languages are C++, Fortran, Java and Python.

To get a flavor of what a high-level language actually looks like, consider an ATM machine where someone wants to make a withdrawal of $100. This amount needs to be compared to the account balance to make sure there are enough funds. The instruction in a high-level computer language would look something like this:

```
x = 100
if balance  x:
print 'Insufficient balance'
else:
 print 'Please take your money'
```

This is not exactly how real people communicate, but it is much easier to follow than a series of 1s and 0s in binary code.

There are a number of advantages to high-level languages. The first advantage is that high-level languages are much closer to the logic of a human language. A high-level language uses a set of rules that dictate how words and symbols can be put together to form a program. Learning a high-level language is not unlike learning another human language - you need to learn vocabulary and grammar so you can make sentences. To learn a programming language, you need to learn commands, syntax and logic, which correspond closely to vocabulary and grammar.

The second advantage is that the code of most high-level languages is portable and the same code can run on different hardware. Both machine code and assembly languages are hardware specific and not portable. This means that the machine code used to run a program on one specific computer needs to be modified to run on another computer. Portable code in a high-level language can run on multiple computer systems without modification. However, modifications to code in high-level languages may be necessary because of the operating system. For example, programs written for Windows typically don't run on a Mac.

A high-level language cannot be understood directly by a computer, and it needs to be translated into machine code. There are two ways to do this, and they are related to how the program is executed: a high-level language can be compiled or interpreted.

A compiler is a computer program that translates a program written in a high-level language to the machine language of a computer. The high-level program is referred to as 'the source code.' A typical computer program processes some type of input data to produce output data. The compiler is used to translate source code into machine code or compiled code. This does

not yet use any of the input data. When the compiled code is executed, referred to as 'running the program,' the program processes the input data to produce the desired output.

When using a compiler, the entire source code needs to be compiled before the program can be executed. The resulting machine code is typically a compiled file, such as a file with an .exe extension. Once you have a compiled file, you can run the program over and over again without having to compile it again. If you have multiple inputs that require processing, you run the compiled code as many times as needed.