33

Learning to code not only prepares young minds for our increasingly tech-driven world, but also allows them to foster creativity, gain problem-solving skills, and improve their overall academic performance.

While it may be difficult to believe that children can learn the intricacies of coding, it's important to remember that computer programming is its own language — and children are experts at absorbing new languages due to their brain plasticity, rapid neural linking abilities, and limited inhibitions.

But, as a teacher or parent who might not be familiar with coding, how do you begin teaching the fundamentals of computer programming? And, what programming language should you focus on?

While there are hundreds of computer programming languages in use today (you might be familiar with names like Java, JavaScript, Python, PHP, Ruby, or C++), most computer programming languages share the same foundational building blocks.

As the foundation of any computer programming language, **variables** act as "containers" that "hold" information. These containers then store this information for later use.

For example, imagine you are visiting the homepage of a website. Once you land on this page, a dialog box pops into view with this simple greeting: "Hi! What's your name?" This dialog box is a variable! In this code, the programmer could name this variable "visitorName." This means that when you type your name into the form and hit submit, your information would be stored in the "visitorName" variable. The programmer could then reference this variable at any time to access the information it contains.

Data structures allow programmers to streamline data collection when a large amount of related information is involved. Let's go back to our "visitorName" variable from above, but imagine the computer programmer needs to store and reference 10 different visitors' names rather than just one.

Rather than creating 10 different variables for each new visitor — which would increase the sheer amount of text in the program and make adding or removing new contacts difficult — the programmer could simply use a data structure to contain all related variables. In this case, the data structure would be a *List*.

With this *List* data structure, the programmer only needs to create one variable rather than 10, which means the code would be much more flexible to change.

34

A **control structure** analyzes variables and selects a direction in which to go determined from the given parameters. For example, when a computer program is running, the code is being read by the computer line by line from top to bottom and (for the most part) left to right.

As the code is being read, the computer will reach a point where it needs to make a "decision" (based on strict rules set by the computer programmer). At this point, the code could do things like jump to a different part of the program, re-run a certain piece of code again, or simply skip a block of code altogether.

Whatever parameters are set by the programmer will affect the code flow. Think of control structures as the directions your program needs to allow it to make choices and execute commands under different conditions.

Just like in the English language, computer programming follows a **syntax** or a set of rules that define particular layouts of letters and symbols. Proper syntax ensures the computer reads and interprets code accurately. For example, let's consider a simple email address and its required syntax.

Email addresses are understood by readers and computers immediately due to their format. Typically, email addresses must consist of a string of letters and numbers, followed by the "@" symbol, and finally a website domain (e.g., bob_smith@companyname.com). This structure is known as the standard email syntax! It's easy to imagine that if the email address were not syntactically correct (company@.comnamebob_smith), computers would not be able to process it.

In a similar fashion, each computer programming language has its own syntax or appropriate order in how code should be written for the program to understand what it is supposed to do.

In the physical world, tools allow workers to perform tasks that would otherwise be extremely difficult (think of how a hammer helps drive a nail into a piece of wood and what this job would be like without tools). Similarly, a **tool** in computer programming is a piece of software that helps programmers write code much faster.

For example, one of the most important tools for computer programmers is an Integrated Development Environment (IDE). An IDE can check the syntax of code for errors, organize files, autocomplete commonly used code, and help you easily navigate through your code. Tools are the final crucial element to code, as they streamline processes and ensure accuracy.

Excerpts from Sphero Team. (2021). *Five Fundamental Coding Concepts Every Kid Should Know*. Sphero.com.

A control structure is a block of programming that analyses variables and chooses a direction in which to go based on given parameters. The term *flow control* details the direction the program takes (which way program control "flows"). Hence it is the basic decision-making process in computing; flow control determines how a computer will respond when given certain conditions and parameters.

Those initial conditions and parameters are called *preconditions*. Preconditions are the state of variables before entering a control structure. Based on those preconditions, the computer runs an *algorithm* (the control structure) to determine what to do. The result is called a *post condition*. Post conditions are the state of variables after the algorithm is run.

Let us analyze flow control by using traffic flow as a model. A vehicle is arriving at an intersection. Thus, the precondition is the vehicle is in motion. Suppose the traffic light at the intersection is red. The control structure must determine the proper course of action to assign to the vehicle.

Precondition: The vehicle is in motion.

Treatments of Information through Control Structures

Is the traffic light green? If so, then the vehicle may stay in motion.
Is the traffic light red? If so, then the vehicle must stop.

End of Treatment

Post condition: The vehicle comes to a stop.

Thus, upon exiting the control structure, the vehicle is stopped.

The **IF-THEN** statement is a simple control that tests whether a condition is true or false. The condition can include a variable, or be a variable. If the variable is an integer 2, it will be true, because any number that is not zero will be true. If the condition is true, then an action occurs. If the condition is false, nothing is done. To illustrate:

IF variable is true

THEN take this course of action.

If the variable indeed holds a value consistent with being true, then the course of action is taken. If the variable is not true, then there is no course of action taken.

IF-THEN statements test for only one action. With an **IF-THEN-ELSE** statement, the control can "look both ways" so to speak, and take a secondary course of action. If the condition is true, then an action occurs. If the condition is false, take an alternate action. To illustrate:

IF variable is true

THEN take this course of action

ELSE call another routine

In this case, if the variable is true, it takes a certain course of action and completely skips the ELSE clause. If the variable is false, the control structure calls a routine and completely skips the THEN clause.

Note that you can combine ELSE's with other IF's, allowing several tests to be made. In an IF-THEN-ELSE IF-THEN-ELSE IF-THEN-ELSE IF-THEN structure, tests will stop as soon as a condition is true. That's why you'd probably want to put the most "likely" test first, for efficiency (Remembering that ELSE's are skipped if the first condition is true, meaning that the remaining portions of the IF-THEN-ELSEIF... would not be processed).

A **WHILE** loop is a process in which a loop is initiated until a condition has been met. This structure is useful when performing iterative instructions to satisfy a certain parameter. To illustrate:

Precondition: variable X is equal to 1

WHILE X is not equal to 9

Add 1 to X

This routine will add 1 to X until X is equal to 9, at which point the control structure will quit and move on to the next instruction. Note that when the structure quits, it will not execute the Add function: when X is equal to 9, it will skip over the clause that is attached to the WHILE. This instruction is useful if a parameter needs to be tested repeatedly before acceptance.

A **DO-WHILE** loop is nearly the exact opposite to a WHILE loop. A WHILE loop initially checks to see if the parameters have been satisfied before executing an instruction. A DO-WHILE loop executes the instruction before checking the parameters. To illustrate:

DO Add 1 to X

WHILE X is not equal 9

As you can see, the example differs from the first illustration, where the DO action is taken before the WHILE. The WHILE is inclusive in the DO. As such, if the WHILE results in a false (X **is equal to** 9), the control structure will break and will not perform another DO. Note that if X is equal to or greater than 9 prior to entering the DO-WHILE loop, then the loop will never terminate.

A **FOR** loop is an extension of a while loop. A for loop usually has three commands. The first is used to set a starting point (like x = 0). The second is the end condition (same as in a while loop) and is run every round. The third is also run every round and is usually used to modify a value used in the condition block.

FOR X is 0. X is less than 10. add 1 to X.

This loop would be run 10 times (x being 0 to 9) so you won't have to think about the X variable in the loop you can just put code there. Here is a while loop that does the same:

X is 0

WHILE X is less than 10

add 1 to X

Some programming languages also have a for each loop which will be useful when working with arrays (and collections). A for each loop is an even more automated while loop that cycles through array's and such. Example :

FOR EACH student **IN** students

give a good mark to student

Excerpts from Wikiversity Team. (2019). *Introduction to Computer Science I*. Saylor.org.