Loops: Repeat after me

Loops are an essential concept in programming that enables developers to repeat a set of instructions multiple times. These instructions can run until a certain condition is met, or until a specified number of iterations are completed. Loops provide a powerful tool for automating repetitive tasks, reducing code duplication, and making programs more efficient.

There are several types of loops that programmers use, including the for loop, while loop, do-while loop, and foreach loop. Each of these loops has a unique way of iterating over a set of values or elements and executing code.

The for loop is the most used loop in programming, which allows a block of code to execute a set number of times based on a range of values. For example, if you want to print out numbers 1 to 10, you could use a for loop to iterate through each number and print it out. For loops are great when you know the conditions or the number of times your loop will be iterating (or repeating). Let us look at an example of a for loop:

Text

Description automatically generated+

You will notice that the for loop contains several things inside of its parenthesis. The first is (int i = 0) this part is the declaration of a variable that will only be used within the loop. If we try to use it outside the body of the loop (i.e. outside the {}) it will lose scope and will no longer be directly usable. The second part is the condition that will be evaluated as the loop continues (i < 5). If this condition is true, the loop will continue only stopping when the condition is false. The final part of the loop is the iteration of the variable (i++). This adds one to the value of our variable i. This is a shorthand version of adding one to a variable in general and can be used to decrement (subtract one i--) or increment as it is now.

The while loop is another type of loop that allows a block of code to execute repeatedly until a certain condition is met. This type of loop is useful when the number of iterations is unknown or dependent on user input. While loops can be incredibly useful and only really have a condition as part of their declaration. Look at the following example of a while loop:

Text

Description automatically generated

The while loop in this instance will loop until a set number is achieved but, we could as an alternative give the loop a simple alternative. Instead of using a number condition, we can give it a Boolean value like so:

A screenshot of a computer

Description automatically generated with medium confidence

The do-while loop is like the while loop but guarantees that the block of code will be executed at least once before the condition is checked. This type of loop is useful when you want to execute a block of code at least once, even if the condition is initially false. Do while version of the while loop above:

Text

Description automatically generated

Finally, the foreach loop is used to iterate over a collection of elements, such as an array or a list. It executes a block of code for each element in the collection, allowing developers to easily work with collections of data.

Something to think about, loops can be nested in the same way that decision structure can be. Meaning that we can place a loop within a loop. Any loop can be placed into any other loop, depending on what kind of behavior we are trying to create with our code. The easiest way to think about a nested loop is to compare it to a clock. A clock contains three loops in essence, two of which are nested loops. The outer most loop is the hour, while the minutes and seconds are the nested loops. The behavior that is exhibited is that for every 60 times the second loops, the minute will loop once. The same can be said of the minute loop and the hour. Nested loops can be set up the same way or in various other ways and can be a very versatile programming tool to add to your toolbox of skills along with loops themselves.