**Loops** : A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

**While loops** instruct your computer to continuously execute your code based on the value of a condition. This works in a similar way to branching if statements. The difference here is that the body of the block can be executed multiple times instead of just once.

**Initializing** : To give an initial value to a variable.

As an IT specialist, while loops can be super helpful. You can use them to keep asking for a username if the one provided isn't valid, or maybe try an operation until it succeeds.

**Anatomy of a While Loop**

A while loop will continuously execute code depending on the value of a condition. It begins with the keyword while, followed by a comparison to be evaluated, then a colon. On the next line is the code block to be executed, indented to the right. Similar to an if statement, the code in the body will only be executed if the comparison is evaluated to be true. What sets a while loop apart, however, is that this code block will keep executing as long as the evaluation statement is true. Once the statement is no longer true, the loop exits and the next line of code will be executed.

One of the most **common errors** is forgetting to initialize variables with the right value. We've all made this mistake when starting to code. Remember how in the earlier examples we initialized our variable x to 0 in one case and to 1 in the other. When we forget to initialize the variable two different things can happen. The first possible outcome and the easiest to catch is that Python might raise an error telling us that we're using a variable we haven't defined.

Whenever you're **writing a loop** check that you're initializing all the variables you want to use before you use them. And don't worry if you don't get it right the first time, we've all been there when learning how to code.

**Infinite Loop** : A loop that keeps executing and never stops

How do you **avoid the most common pitfalls when writing while loops?** First, remember to initialize your variables, and second, check that your loops won't run forever.

**Common Errors in while Loops:**

If you get an error message on a loop or it appears to hang, your debugging checklist should include the following checks:

* Failure to initialize variables. Make sure all the variables used in the loop’s condition are initialized before the loop.
* Unintended infinite loops. Make sure that the body of the loop modifies the variables used in the condition, so that the loop will eventually end for all possible values of the variables. You can often prevent an infinite loop by using the break keyword or by adding end criteria to the condition part of the while loop.

**while Loop Terms**

* while loop - Tells the computer to execute a set of instructions while a specified condition is True. In other words, while loops keep executing the same group of instructions until the condition becomes False.
* infinite loop - Missing a method for exiting the loop, causing the loop to run forever.
* break - A keyword that can be used to end a loop at a specific point.

**Math Concepts on the Practice Quiz**

The coding problems on the upcoming practice quiz will involve a few math concepts. Don’t worry if you are rusty on math. You will have plenty of support with these concepts on the quiz. The following is a quick overview of the math terms you will encounter on the quiz:

* **prime numbers** - Integers that have only two factors, which are the number itself multiplied by 1. The lowest prime number is 2.
* **prime factors** - Prime numbers that are factors of an integer. For example, the prime numbers 2 and 5 are the prime factors of the number 10 (2x5=10). The prime factors of an integer will not produce a remainder when used to divide that integer.
* **divisor** - A number (denominator) that is used to divide another number (numerator). For example, if the number 10 is divided by 5, the number 5 is the divisor.
* **sum of all divisors of a number** - The result of adding all of the divisors of a number together.
* **multiplication table** - An integer multiplied by a series of numbers and their results formatted as a table or a list.

**For Loops :** Iterates over a sequence of values

There are **two important thing**s I want to call out about this **range function**. First, in Python and a lot of other programming languages, a range of numbers will **start with the value 0** by default. Second, the list of numbers generated will be **one less than the given value.**

Some examples of **sequences that we can iterate** are the files in a directory, the lines in a file, the processes running on a machine. And there's a bunch of others. So as an IT specialist, you'll use for loops to automate tons of stuff. For example, you might use them to copy files to machines, process the contents of files, automatically install software, and a lot more.

If you're wondering when you **should use for loops and when you should use while loops**, there's a way to tell. **Use for loops when** there's a sequence of elements that you want to iterate. **Use while loops when** you want to repeat an action until a condition changes. And if whatever you're trying to do can be done with either for or while loops, just use whichever one's your favorite.

**The range() function**

The **range()** function uses a set of indices that point to integer values, which start at the number 0. The numeric values 0, 1, 2, 3, 4 correlate to ordinal index positions 1st, 2nd, 3rd, 4th, 5th. So, when a range call to the 5th index position is made using range(5) the index is pointing to the numeric value of 4. **The range()** function can take up to three parameters: **range(start, stop, step)**

**Start**

The first item in the range() function parameters is the starting position of the range. The default is the first index position, which points to the numeric value 0. This value is included in the range.

**Stop**

The second item in the range() function parameters is the ending position of the range. There is no default index position, so this index number must be given to the range() parameters. For example, the line for n in range(4) will loop 4 times with the n variable starting at 0 and looping 4 index positions: 0, 1, 2, 3. As you can see, range(4) (meaning index position 4) ends at the numeric value 3. In Python, this structure may be phrased as “the end-of-range value is excluded from the range.” In order to include the value 4 in range(4), the syntax can be written as range(4+1) or range(5). Both of these ranges will produce the numeric values 0, 1, 2, 3, 4.

**Step**

The third item in the range() function parameters is the incremental step value. The default increment is +1. The default value can be overridden with any valid increment. However, note that the loop will still end at the end-of-range index position, regardless of the incremental value. For example, if you have a loop with the range: for n in range(1, 5, 6), the range will only produce the numeric value 1. This is because the incremental value of 6 exceeded the ending point of the range.

**Nested For Loops** : Turning this into code means that we'd need to write two for loops, one inside the other.

**For loops are best when** you want to iterate over a known sequence of elements but when you want to **operate while a certain condition is true**, while loops are the best choice.

**For Loops Vs. While Loops**

**For loops** and **while loops** share several characteristics. Both loops can be used with a variety of data types, both can be nested, and both can be used with the keywords **break** and **continue**. However, there are important differences between the two types of loops:

* **while loops** are used when a segment of code needs to execute repeatedly while a condition is true
* **for loops** iterate over a sequence of elements, executing the body of the loop for each element in the sequence

**Recursion :** The Repeates application of the same procedure to a smaller problem

Recursion lets us tackle complex problems by reducing the problem to a simpler one. In programmiong, recursion is a way of doing a repetitive task by having a function call itself.