While loop::

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.

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.

x+=1 <==> x=x+1

whenever you're writing a loop check that you're initializing all the variables you want to use before you use them.

WARNING!!

You'll want to watch out for a common mistake: forgetting to initialize variables. If you try to use a variable without first initializing it, you'll run into a NameError. This is the Python interpreter catching the mistake and telling you that you’re using an undefined variable. The fix is pretty simple: initialize the variable by assigning the variable a value before you use it.

Another common mistake to watch out for that can be a little trickier to spot is forgetting to initialize variables with the correct value. If you use a variable earlier in your code and then reuse it later in a loop without first setting the value to something you want, your code may wind up doing something you didn't expect. Don't forget to initialize your variables before using them!

You can use logical operators(and, or) within a while condition.

Infinite loop::

#an infinite loop  
x=0  
while x%2==0:  
 x=x/2  
 print(x)

This program keeps running and printing x=0 forever.

While you need to watch out for infinite loops, they are not always a bad thing. Sometimes you actually want your program to execute continuously until some external condition is met. If you've used the ping utility on Linux or macOS system, or ping-t on a Windows system, you've seen an infinite loop in action. This tool will keep sending packets and printing the results to the terminal unless you send it the interrupt signal, usually pressing Ctrl+C. If you were looking at the program source code, you'll see that it uses an infinite loop to do this with a block of code with instructions to keep sending the packets forever. One thing to call out is it should always be possible to break the loop by sending a certain signal. In the ping example, that signal is the user pressing Ctrl+C. In other cases, it could be that the user pressed the button on a graphical application, or that another program sent a specific signal, or even that a time limit was reached.

Print a message once the condition is false.

i = 1

while i < 6:

print(i)

i += 1

else:

print("i is no longer less than 6")

A for loop iterates over a sequence of values.

The list of numbers generated by the range() will always be one less than the given value

Usages of for loop

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 oops, 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.

a while loop is great for performing an action over and over until a condition has changed. A for loop works well when you want to iterate over a sequence of elements.

Common Errors in for loop::

for loops iterate over sequences. The interpreter will refuse to iterate over a single element.

for x in 25:  
 print(x)

output::

*TypeError: 'int' object is not iterable*

There are two solutions to this problem, depending on what we're trying to do. If we want to go from zero to 25, then we use the range function. But if we're trying to iterate over a list that has 25 as the only element, then it needs to be a list and that means writing 25 between square brackets.

**Python end parameter in print()**

It simply appends the next line to the previous line.

By default python’s print() function ends with a newline. Python’s print() function comes with a parameter called ‘end’. By default, the value of this parameter is ‘\n’, i.e. the new line character. You can end a print statement with any character/string using this parameter.

*print("Welcome to" , end = ' ')*

*print("GeeksforGeeks", end = ' ')*

Output :

Welcome to GeeksforGeeks

One more program to demonstrate working of end parameter.

*print("Python" , end = '@')*

*print("GeeksforGeeks")*

Output :

Python@GeeksforGeeks

**Recursion in IT ::**

A recursive function must include a recursive case and base case. The recursive case calls the function again, with a different value. The base case returns a value without calling the same function

Let's say that you need to write a tool that goes through a bunch of directories in your computer and calculates how many files are contained in each. When listing the files inside a directory, you might find subdirectories inside them and you'd want to count the files in those subdirectories as well. This is a great time to use recursion. The base case would be a directory with no subdirectories. For this case, the function would just return the amount of files. The recursive case would be calling the recursive function for each of the contained subdirectories. The return value of a given function call would be the sum of all the files in that directory plus all the files in the contained subdirectories. A directory of files that can contain other directories is an example of a recursive structure.

Another IT-focused example of a recursive structure is anything that deals with groups of users that can contain other groups. We see this situation a lot when using administrative tools like active directory or LDAP. Say your group management software allows you to create groups that have both users and other groups as their members. And you want to list all human users that are part of a given group. Here you would use a recursive function to go through the groups. The base case would be a group that only includes users listing all of them. The recursive case would mean going through all the groups contained listing all the users in them and then listing any users contained in the current group.

In Python by default, you can call a recursive function 1,000 times until you reach the limit. That's fine for things like subdirectories or user groups that aren't thousands of levels deep. But it might not be enough for mathematical functions like finding factorial.