Introduction to Loops

by Sophia

WHAT'S COVERED

In this lesson, we will learn about the basics of iteration and the properties and uses of while and for loops. Specifically, this lesson covers:

1. Iteration

We briefly discussed iteration a few lessons ago when we learned about the `iter()` and `next()` functions, and used those functions to move through an iterable object. With the topic of loops, however, it's a great time to revisit iteration, which basically means a repetition of a process. Computers are often used to automate repetitive tasks. Repeating identical or similar tasks without making errors is something that computers do well, and people do poorly. Because iteration is so common, Python provides several language features to make it easier.

Loops allow us to repeat code multiple times. This repetitive process is called iteration in programming. A programming feature that implements iteration well is called a loop.

Two types of iteration

In programming, there are two types of iteration with loops: indefinite and definite iteration.

With **indefinite iteration**, we don't specify the number of times that the loop is meant to run in advance. Instead, the loop would just run repeatedly as long as a condition is met, which is similar to a conditional statement that is checked each time. For example, think of a video game menu that keeps repeating the selections until the user quits the game by choosing the option to exit. In Python, indefinite loops are typically created using the `while` loop.

With **definite iteration**, the number of times the block of code runs should be explicitly defined when the loop actually starts. Typically, this is implemented using the `for` loop. The `for` loop has a specific start and

endpoint.

Often, we use a `for` or `while` loop to go through a list of items/elements or the contents of a file when we are looking for something such as the largest or smallest value of the data. We'll be looking at how to work with files using loops in a later lesson.

The following are generalizations about loops:

- Initializing one or more variables before the loop starts.
    - This will set up the conditions we wish to check on or an element/value we are looking for. It doesn't make sense to create a loop to execute if there is nothing to iterate through.
- Performing some computation on each statement in the body of the loop (the looping block of code), or possibly changing the variables in the body of the loop.
    - Again, what is expected to happen during the loop? Are we looking for an element/value, changing variables, running functions, or perhaps counting a value?
- Looking at the resulting variables when the loop completes.
    - The loop we created should have done something to a data collection type or variable(s). Did it do what we expected it to?

TERMS TO KNOW

Indefinite Iteration
A loop that is not specified in advance on how many times to be run; it repeats as long as a condition is met. In Python, indefinite loops are typically created using the `while` loop.

Definite Iteration
With definite iteration, the number of times the block of code runs should be explicitly defined when the loop actually starts. Typically, this is implemented using the `for` loop. The `for` loop has a specific start and endpoint.

---

2. Basics of a while Loop

The **while loop** is one of the most commonly used loops. A `while` loop continually evaluates a condition looking for a True or False value. It keeps going (looping) as long as the evaluated condition is True. Let's first look at the structure of the `while` loop.

EXAMPLE
The format of a `while` loop looks like the following:

```
while <expression>:
  <statement(s)>
```

Note: In the example above, the <expression> and <statement(s)> terms and outside arrows are just for information purposes; these are not keywords or actual code. These are just to explain what goes into each of the parts of a `while` loop.

First, we'll discuss the <statement(s)> term. The <statement(s)> represent the block of code that should be executed repeatedly. This is also called the **body of the loop**. This is notated in the same way as a conditional statement, with an indent. We will see that all of the iteration features like the `while` and `for` loop use the same style of indentation as we previously used with the if/else/elif conditional statements in Python to define the code blocks.

The <expression> term is typically based on one or more variables that are initialized outside of the loop and then modified within the body of the loop. The <expression> represents the evaluated condition. Python is looking for a boolean True or False value from this condition.

How does a while loop work?

During execution (runtime) when a `while` loop is reached, first, the program will evaluate the <expression> (looking at what conditions are in the expression). If the result is True, the body of the loop will execute. Once all of the statements in the body of the loop are executed, the <expression> is checked again. If it is still True, the body of the loop executes again. This is where the term "loop" comes from, because the last statement in the body of the loop, loops back around to the top. We call each time we execute the body of the loop an iteration. This process keeps running until the <expression> becomes False. When that occurs, the program moves on to the first statement that is outside of the body of the loop.

EXAMPLE

Let's look at a simple example to demonstrate how this works.

```
myNum = 5
while myNum > 0:
      myNum = myNum - 1
      print(myNum)
print("Our loop is done")
```

In the code snippet above, the code is doing the following (table view):

Inside or outside the `while` loop	Step	Code	What's happening	`myNum` value during each iteration (loop) and output
Outside	1	`myNum = 5`	We initialize the variable `myNum` with a numeric value of 5.	n/a

Inside	2	`while myNum > 0:`	**While Loop** The `while` loop is initiated and its <expression> is checking if `myNum` is greater than 0; and if True, do the following statements in the body of the loop.	Expression check 1: myNum=5 (True) Expression check 2: myNum=4 (True) Expression check 3: myNum=3 (True) Expression check 4: myNum=2 (True) Expression check 5: myNum=1 (True) Expression check 6: myNum=0 (False, 0 is not greater than 0 so break out of the loop and continue on to the next statement outside of loop lines of code (step 3))
Inside	2A	`myNum = myNum - 1`	**Start Body of the Loop** Subtract 1 from `myNum`.	Loop 1: myNum = 4 Loop 2: myNum = 3 Loop 3: myNum = 2 Loop 4: myNum = 1 Loop 5: myNum = 0
Inside	2B	`print(myNum)`	Print `myNum` to screen.	Loop 1: Output = 4 Loop 2: Output = 3 Loop 3: Output = 2 Loop 4: Output = 1 Loop 5: Output = 0
Inside	2C		**End Body of Loop** Go back to check the `while` loop	

			expression (step 2) with updated `myNum`.	
Outside	3	print("Our loop is done")	Print “Our loop is done”.	Output = “Our loop is done”

And here is what is happening in paragraph summary:

First, `myNum` is initialized to 5 outside of the loop. Then we get to the `while` loop. As such, the expression `myNum > 0` is tested. Since `myNum` is equal to 5 and that is greater than 0, the expression returns True. So, the body of the loop is executed. On the third line, `myNum` is decremented by 1 to 4. Then on the next line, the value of `myNum` is printed, which is 4.

Since the body of the loop is finished, the program returns back to the top of the loop on line 2 and the expression is evaluated again. With `myNum` now being set to 4, this still returns True, so the body of the loop is executed again and 3 is output. This continues on until `myNum` becomes 0 within the body of the loop. At that point, when we evaluate the expression, `myNum` is no longer greater than 0. As such, it returns False, so the loop terminates and goes to the first statement after the loop, which is our line that outputs “Our loop is done”.

If we run this loop, here is the output.

```
4
3
2
1
0
Our loop is done
```

It looks to be what was expected before the condition of the expression was met with a False check.

Since the expression in the `while` loop is tested first, it’s possible that it could have been False to begin with, which means the body of the loop would never have run at all.

KEY CONCEPT

Iteration Variable

The body of the loop should change the value of one or more variables so that eventually the condition becomes false and the loop terminates. The variable that changes each time the loop executes and controls when the loop finishes is called the iteration variable. If there is no **iteration variable**, the loop will repeat forever, resulting in an infinite loop. An **infinite loop** is a loop in which the terminating condition is never satisfied or for which there is no terminating condition.

For our previous example, `myNum` acted as the iteration variable. With each iteration, its value was changed (1 was subtracted from value). This ensured that at some point, the condition would be evaluated as False, thus ending the loop.

EXAMPLE

In the example below, we're using the same code but myNum is first set to 0. Since 0 > 0 returns False, the body of the loop never executes.

```
myNum = 0
while myNum > 0:
      myNum = myNum - 1
      print(myNum)
print("Our loop did nothing")
```

The output on this loop.

```
Our loop did nothing
```

We changed the final output string to let us know nothing happened. During runtime, the while loop condition was checked, proven to be False, and jumped out of the loop to the next line of code, which happened to be the final `print()` function.

while

The `while` loop is one of the most commonly used loops. It keeps going as long as some condition is true.

Body of the Loop

The body of the loop represents the indented block of code that should be executed repeatedly within the loop during each loop iteration.

Iteration Variable

The variable that changes each time the loop executes and controls when the loop finishes.

Infinite Loop

An infinite loop is a loop in which the terminating condition is never satisfied or for which there is no terminating condition.

---

3. Basics of a for Loop

In Python, definite iteration loops are generally called **for** loops. There are a few common types of `for` loops. Python's for loop is a data collection-based iteration (meaning it loops through iterable objects such as lists, sets, tuples, dictionaries, and even strings).

↷ EXAMPLE

```
for <variable> in <iterable>:
  <statement(s)>
```

Note: Again, in the example above, the <variable>, <iterable>, and <statement(s)> terms with outside arrows are just for information purposes. These are not keywords or actual code. These are just to explain what goes into each of these parts of a `for` loop.

The <variable> is initially declared and used to refer to each of the elements in the iterable object (such as a list).

The <iterable> is an object that is capable of going through its members (elements) one at a time. This should be familiar from when we covered the list, tuple, set, and dictionary. They are all iterable objects.

The <statement(s)> are the statements that will execute for each element in the <iterable>. The <variable> refers to the next element in <iterable> every time that the loop is executed.

EXAMPLE
For example:

```
productList = ("card", "paper", "glue", "pencil")
for product in productList:
  print(product)
```

In this example, the <iterable> is the list called `productList` and is created outside of the loop. The <variable> is a variable called `product`. Each time through the loop, `product` refers to the next element of the list `productList`, so the `print(product)` displays each element's value one at a time.

```
card
paper
glue
pencil
```

So, after 4 iterations (loops), the `for` loop covered all elements of the list `productList` and output those elements to the screen.

The `for` loop can also loop using a `range()` function with start and end values. The format looks like this.

EXAMPLE

```
for counter in range(0,5):
    print("Counter is set to:",counter)
```

The print function is in the body of the loop. Here, we have a loop that executes 5 times. It does this by using the `range()` function. The **range() function** takes multiple parameters, the starting number (0 by default), and the ending number. This function will increment by 1 (by default) and then stop before a designated number, so we will get a sequence of numbers between the starting and ending numbers. The function can take an optional third parameter to change the step value (increment or decrement by a specific value other than the default of 1). We create the variable counter and first set it to 0 on the first iteration of the loop, then 1, and so on until it reaches 5. Once the variable counter reaches 5, it exits the `for` loop without executing the body of the loop since the condition was met.

Here is the output.

```
Counter is set to: 0
Counter is set to: 1
Counter is set to: 2
Counter is set to: 3
Counter is set to: 4
```

KEY CONCEPT

Notice that it does not execute or output 5, as the number 5 is the exit value. If we wanted to include 5 as part of the output (and have the loop execute 6 times), we should put in 6 as the final value in the `range()` function parameter.

`for` loops usually start from 0, so we could also remove the 0 in the `range()` function.

EXAMPLE

```
for counter in range(5):
   print("Counter is set to:",counter)
```

The output of the loop:

```
Counter is set to: 0
Counter is set to: 1
Counter is set to: 2
Counter is set to: 3
Counter is set to: 4
```

See? We get the same output without adding the start value (as long as we wanted the start value to be the first element at the 0 index position).

If we wanted the loop to start at a different number such as 1, we would have to specify that in the `range()` function.

EXAMPLE

```
for counter in range(1,5):
   print("Counter is set to:",counter)
```

The output of the loop:

```
Counter is set to: 1
Counter is set to: 2
```

```
Counter is set to: 3
Counter is set to: 4
```

This time the loop did not capture the 0 index since we gave the start value of 1.

DID YOU KNOW

We can also use negative numbers in the `range()` function.

Let's say we used -2 as the starting number and 3 as the ending number.

EXAMPLE

```
for counter in range(-2,3):
    print("Counter is set to:",counter)
```

The output of the loop:

```
Counter is set to: -2
Counter is set to: -1
Counter is set to: 0
Counter is set to: 1
Counter is set to: 2
```

Here, our output counter is -2 to 2. So, you have the ability to start with a negative number.

Also, you can use the optional third parameter to increment differently than the default of 1. This incremental parameter can be positive or negative too. For example:

EXAMPLE

```
for counter in range(10,1,-2):
    print("Counter is set to:",counter)
```

With the -2 as the third parameter, we are moving through in increments of -2 now.

```
Counter is set to: 10
Counter is set to: 8
Counter is set to: 6
Counter is set to: 4
Counter is set to: 2
```

We could also use a `for` loop through a string, as it will repeat once for each character in the string. The variable will contain one character from the string with each pass of the loop going from left to right.

EXAMPLE

```
for char in "Python":
   print(char)
```

The output shows the string Python.

```
P
y
t
h
o
n
```

Notice that each character in the string is output one at a time. Here is another example.

EXAMPLE

```
for pet in ("dog","cat","fish"):
  print(pet)
```

The output shows each element one at a time.

```
dog
cat
fish
```

TERMS TO KNOW

for

In Python, definite iteration loops are generally called `for` loops. Python's `for` loop is a data collection-based iteration (loops through iterable objects such lists, sets, tuples, dictionaries, and even strings).

range()

The `range()` function takes multiple parameters, the starting number (0 by default), and the ending number. This function will increment by 1 (by default) and then stop before a designated number. The function can take an optional third parameter to change the step value (increment or decrement by a specific value other than the default of 1).

SUMMARY

In this lesson, we reviewed the basics of **iteration** and defined that indefinite iteration is a loop that runs as long as a condition is True and will break from the loop when that condition becomes False. We also defined definite iteration, which relies on defined start and stop values. We learned the **basic**

properties of a while loop and that these loops are generally used for indefinite conditions where the number of times the loop should be run is not known. Finally, we discussed the **basics of a for loop** and that they are used for definite iterations where we know ahead of time how many times the loop should run.

Best of luck in your learning!

Source: THIS CONTENT AND SUPPLEMENTAL MATERIAL HAS BEEN ADAPTED FROM “PYTHON FOR EVERYBODY” BY DR. CHARLES R. SEVERANCE ACCESS FOR FREE AT www.py4e.com/html3/ LICENSE: CREATIVE COMMONS ATTRIBUTION 3.0 UNPORTED.

TERMS TO KNOW

Body of the Loop
The body of the loop represents the indented block of code that should be executed repeatedly within the loop during each loop iteration.

Definite Iteration
With definite iteration, the number of times the block of code runs should be explicitly defined when the loop actually starts. Typically, this is implemented using the for loop. The for loop has a specific start and endpoint.

Indefinite Iteration
A loop that is not specified in advance on how many times to be run; it repeats as long as a condition is met. In Python, indefinite loops are typically created using the while loop.

Infinite Loop
An infinite loop is a loop in which the terminating condition is never satisfied or for which there is no terminating condition.

Iteration Variable
The variable that changes each time the loop executes and controls when the loop finishes.

for
In Python, definite iteration loops are generally called for loops. Python’s for loop is a data collection-based iteration (loops through iterable objects such lists, sets, tuples, dictionaries, and even strings).

range()
The range() function takes multiple parameters, the starting number (0 by default), and the ending number. This function will increment by 1 (by default) and then stop before a designated number. The function can take an optional third parameter to change the step value (increment or decrement by a specific value other than the default of 1).

while

The while loop is one of the most commonly used loops. It keeps going as long as some condition is true.