

04-while Loops

October 31, 2020

Coursework delivered by: Alison Mukoma

Copyright: Evelyn Hone College cc DevsBranch.

1 while Loops

The while statement in Python is one of most general ways to perform iteration. A while statement will repeatedly execute a single statement or group of statements as long as the condition is true. The reason it is called a 'loop' is because the code statements are looped through over and over again until the condition is no longer met.

The general format of a while loop is:

```
while test:
    code statements
else:
    final code statements
```

Let's look at a few simple while loops in action.

```
[1]: x = 0

while x < 10:
    print('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')
    x+=1
```

```
x is currently: 0
x is still less than 10, adding 1 to x
x is currently: 1
x is still less than 10, adding 1 to x
x is currently: 2
x is still less than 10, adding 1 to x
x is currently: 3
x is still less than 10, adding 1 to x
x is currently: 4
x is still less than 10, adding 1 to x
x is currently: 5
```

```
x is still less than 10, adding 1 to x
x is currently: 6
x is still less than 10, adding 1 to x
x is currently: 7
x is still less than 10, adding 1 to x
x is currently: 8
x is still less than 10, adding 1 to x
x is currently: 9
x is still less than 10, adding 1 to x
```

Notice how many times the print statements occurred and how the while loop kept going until the True condition was met, which occurred once `x==10`. It's important to note that once this occurred the code stopped. Let's see how we could add an else statement:

```
[2]: x = 0

while x < 10:
    print('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')
    x+=1

else:
    print('All Done!')
```

```
x is currently: 0
x is still less than 10, adding 1 to x
x is currently: 1
x is still less than 10, adding 1 to x
x is currently: 2
x is still less than 10, adding 1 to x
x is currently: 3
x is still less than 10, adding 1 to x
x is currently: 4
x is still less than 10, adding 1 to x
x is currently: 5
x is still less than 10, adding 1 to x
x is currently: 6
x is still less than 10, adding 1 to x
x is currently: 7
x is still less than 10, adding 1 to x
x is currently: 8
x is still less than 10, adding 1 to x
x is currently: 9
x is still less than 10, adding 1 to x
All Done!
```

2 break, continue, pass

We can use break, continue, and pass statements in our loops to add additional functionality for various cases. The three statements are defined by:

break: Breaks out of the current closest enclosing loop.

continue: Goes to the top of the closest enclosing loop.

pass: Does nothing at all.

Thinking about break and continue statements, the general format of the while loop looks like this:

```
while test:
    code statement
    if test:
        break
    if test:
        continue
else:
```

break and continue statements can appear anywhere inside the loop's body, but we will usually put them further nested in conjunction with an if statement to perform an action based on some condition.

Let's go ahead and look at some examples!

```
[3]: x = 0

while x < 10:
    print('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')
    x+=1
    if x==3:
        print('x==3')
    else:
        print('continuing...')
        continue
```

```
x is currently:  0
 x is still less than 10, adding 1 to x
continuing...
x is currently:  1
 x is still less than 10, adding 1 to x
continuing...
x is currently:  2
 x is still less than 10, adding 1 to x
x==3
x is currently:  3
 x is still less than 10, adding 1 to x
continuing...
x is currently:  4
 x is still less than 10, adding 1 to x
```

```

continuing...
x is currently: 5
  x is still less than 10, adding 1 to x
continuing...
x is currently: 6
  x is still less than 10, adding 1 to x
continuing...
x is currently: 7
  x is still less than 10, adding 1 to x
continuing...
x is currently: 8
  x is still less than 10, adding 1 to x
continuing...
x is currently: 9
  x is still less than 10, adding 1 to x
continuing...

```

Note how we have a printed statement when $x==3$, and a continue being printed out as we continue through the outer while loop. Let's put in a break once $x == 3$ and see if the result makes sense:

```

[4]: x = 0

while x < 10:
    print('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')
    x+=1
    if x==3:
        print('Breaking because x==3')
        break
    else:
        print('continuing...')
        continue

```

```

x is currently: 0
  x is still less than 10, adding 1 to x
continuing...
x is currently: 1
  x is still less than 10, adding 1 to x
continuing...
x is currently: 2
  x is still less than 10, adding 1 to x
Breaking because x==3

```

Note how the other else statement wasn't reached and continuing was never printed!

After these brief but simple examples, you should feel comfortable using while statements in your code.

A word of caution however! It is possible to create an infinitely running loop with while statements. For example:

```
[ ]: # DO NOT RUN THIS CODE!!!!  
while True:  
    print("I'm stuck in an infinite loop!")
```

A quick note: If you *did* run the above code, you might want to restart your computer

```
[ ]:
```