While Loop

The while-loop has more flexibility, looping until a boolean test is False. The earlier for-loop is very handy to loop over a collection, but that collection needs to be known ahead of time.

While Loop

The while-loop uses a boolean test expression to control the run of the body lines. The for-loop is great of looping over a collection. The while-loop is more general, providing enough control for any sort of looping, without requiring a collection to loop over.

While Loop Syntax

The while-loop syntax has 4 parts: while, boolean test expression, colon, indented body lines:

```
while test:
    indented body lines
```

While Operation: Check the boolean test expression, if it is True, run all the "body" lines inside the loop from top to bottom. Then loop back to the top, check the test again, and so on. When the test is False, exit the loop, running continues on the first line after the body lines.

Here is a while loop to print the numbers 0, 1, 2, ... 9 (there are easier ways to do this, but here we're just trying to show the parts of the loop).

```
i = 0
while i < 10:
    print(i)
    i = i + 1
print('All done')

0
1
2
3
```

```
4
5
6
7
8
9
All done
```

Very often the last line of the while body has an "increment" role, such as the `i = i + 1` line above. The test at the top of the loop checks that variable. It's important that on every iteration, the loop advances that variable one step towards the ultimate end of the loop.

While Zero Iterations OK

Just as with the for-loop, a while-loop can iterate zero times. That happens if the boolean test is False the very fist time it is checked, like this:

```
i = 99
while i < 10:
    print(i)
    i += 1
print('All done')

# (zero iterations - no numbers print at all)
All done
```

Infinite Loop Bug

With a while-loop, it's possible to accidentally write a loop that never exits. In that case, the while just loops and loops but never makes the test False to exit. As the loop runs and runs, the fans on your laptop may spin up as CPU heats up with this high number of lines running without pause.

Here's an infinite loop example caused by a typical looking bug — the variable `i` accidentally stays at the value 1 and the loop just goes forever.

```
i = 0
while i < 10:    # BUG infinite loop
    print(i)
    i = i * 1
print('All done')
```

Don't Forget the Last "Increment" Line

Another easy infinite loop bug is forgetting the `i = i + 1` line entirely, so the variable never advances and the loop never exits. Since the more

commonly used for-loop automates the increment step for us, we don't quite have the muscle memory to remember it when writing a while-loop.

Do Not Write == True

Suppose there is some function `foo()` and you want a while loop to run so long as it returns True. Do not write this

```
while foo() == True:    # NO not this way
    ...
```

It's better style to write it the following way, letting the while itself evaluate the True/False of the test:

```
while foo():            # YES this way
    ...
```

While break and continue

The `break` and `continue` directives work the same as in the for-loop (See also for loop).

The `break` provides an extra way to exit the loop. Typically the `break` is nested inside an if. The if/break can be positioned anywhere in the loop vs. the while/test can only be at the top of the loop. Here is the previous example with a `break` added to leave the loop if the number is 6:

```
i = 0
while i < 10:
    print(i)
    if i == 6:
        break
    i = i + 1
print('All done')

0
1
2
3
4
5
6
All done
```

While String break

Here is a more realistic example of `break` looping over a string. The while loop goes through the index numbers in the usual way 0, 1, 2, .. len-1. In the

loop, an if statement checks for a digit, and breaks out of the loop when found. If there is no digit, the while loop will exit when it reaches the end of the string as usual.

```
s = 'abc123'

i = 0
while i < len(s):
    print(i, s[i])
    if s[i].isdigit():
        print('digit at', i)
        break
    i += 1
print('All done')
```

```
For s = 'abc123' output:
0 a
1 b
2 c
3 1
digit at 3
All done
```

While True

Given that we have if/break to get out of the loop, it's possible to get rid of the test at the top of the loop entirely, relying on if/break to end the loop. We do this by writing the loop as `while True:...`

Here is an example that uses while/True to go through the numbers 0..9. This not the best way to generate those numbers; it just shows how if/break can serve instead of a while/test at the top of the loop.

```
i = 0
while True:
    print(i)
    i += 1
    if i >= 10:
        break
```

```
0
1
2
3
4
5
6
7
```

```
8
9
```


Copyright 2020 Nick Parlante