Google

Reading: Loops, break, and continue statements

You've learned about while loops in Python and have explored some examples. While loops are useful because they allow you to perform an action or evaluation repeatedly until a given condition or requirement is met, and then they stop. This is an important process in computer programming, not just in Python, but in most other languages too. Data professionals use while loops to process data, so it's important for you to familiarize yourself with them as you grow your skills. This reading is a review of the fundamental concepts of while loops.

While loop syntax

A while loop is a control structure that allows you to repeatedly execute a block of code for as long as a certain condition is true.

Note: The following code block is not interactive.

The basic syntax of a while loop is as follows:

```
while condition:
  # Code block to execute
```

The **condition** is a Boolean expression that is evaluated at the beginning of each iteration of the loop. If the condition is true, the code block executes. After the code block executes, the condition is evaluated again. This process continues until the condition is false, at which point the loop terminates and the program continues with the next statement after the loop.

Here is an example of a basic while loop:

```
x = 1
while x < 100:
  print(x)
  x = x*2
```

Output:

```
1
2
4
8
```

```
16
32
64
```

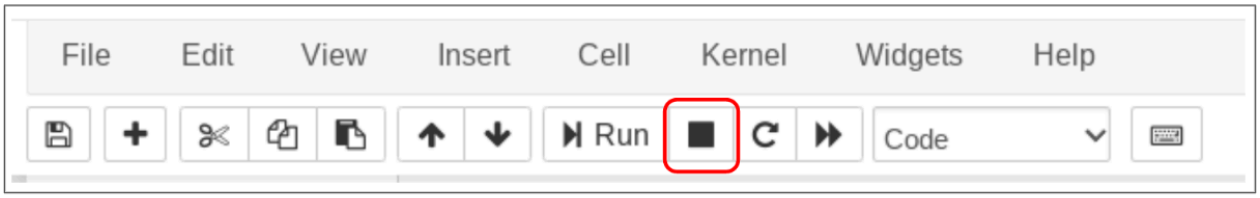
In this example, **x** equals one when the loop begins. Because **x** is less than 100, the program prints the value of **x**, then multiplies **x** by two. Then the condition is checked again, and because it is still **True**, the code inside the loop executes again. This process continues until **x** becomes 128, at which point the condition becomes **False** and the loop terminates.

Infinite loops

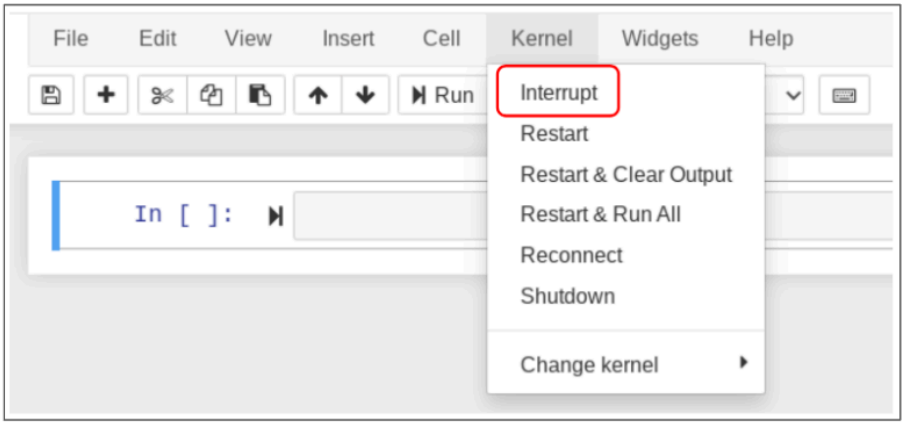
Be careful with while loops because if you make a mistake with your logic or syntax, it could result in an infinite loop that never terminates. In the previous example, if x = x*2 were accidentally not indented to be in the body of the while loop, the loop would reach the print statement and cycle back to check the conditional statement, which of course would still be true because the value of x would never change from one.

If you get stuck in an infinite loop, don't worry. You can break out of it by interrupting the kernel. There are several ways to do this:

1. Use the stop button in the menu at the top of the notebook.



2. Go to Kernel in the menu bar at the top of the notebook and select Interrupt from the drop-down menu.



3. While in command mode, press i twice.

break & continue

It is possible to end a loop even if the conditional statement is still true. To do this, use a **break** statement.

Here's an example:

```
x = 1
i = 0
while x < 100:
  if i == 5:
    break
  print(i, x)
  x = x*2
  i += 1
```

Output:

```
0 1
1 2
2 4
3 8
4 16
```

In this example, there is a variable **i** that acts as a counter. For each iteration of the loop, the program:

1. Checks if **x** is less than 100.
2. If it is, then the program checks if **i** equals five.
3. If it does, the loop terminates because of the break statement. Otherwise, it prints the values of both **i** and **x**, doubles the value of **x**, and increments the value of **i** by one.
4. Repeats until **x** ≥ 100 or **i** = 5. In this case, the loop breaks when i becomes 5.

It's also possible to skip an iteration of the loop without executing the rest of the code inside the loop for the current iteration. To do this, use a **continue** statement.

Here's an example:

```
i = 0
while i < 10:
  if i % 3 != 0:
    print(i)
    i += 1
    continue
```

```
    i += 1
```

Output:

```
1
2
4
5
7
8
```

This example is a loop that prints all the numbers from zero through 9 that are not divisible by three. For each iteration of the loop, the program:

1. Checks if **i** is less than 10.
2. If it is, then the program uses the modulo operator to check if **i** is evenly divisible by three.
3. If it is not, then the program prints **i**, increments the value of **i** by one, *and then cycles back to the beginning* to check that **i** is less than 10. This happens because of the continue statement. The final **i += 1** does not execute, thus avoiding a double incrementation of **i**.
4. But if step 2 evaluates **i** as evenly divisible by three, nothing in the if block executes (so there's no print statement) and **i** is incremented by one.
5. Repeats until **i** becomes 10.

Key takeaways

A **while** loop allows you to repeatedly execute a block of code while a certain condition is true. You can use the **break** statement to exit the loop prematurely, and the **continue** statement to skip to the next iteration of the loop without executing the rest of the code in the current iteration.
