Lesson 2 – Loops and Conditional Statements

Author: Jake Palczewski, CCIE #63320
Last Updated: November 29, 2020

Table of Contents

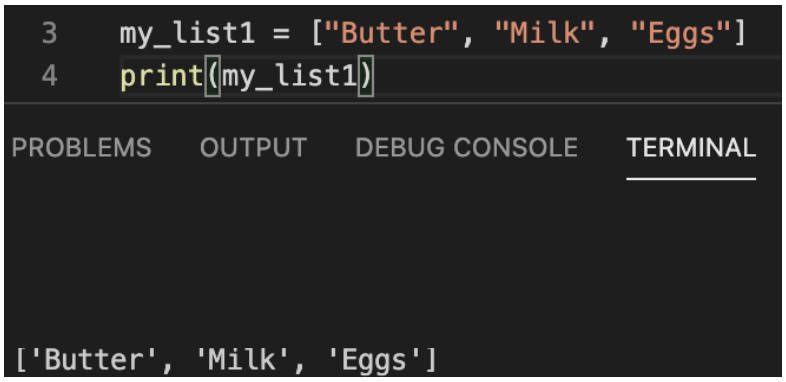
Introduction

It's time to discuss loops and conditional statements, which make use of variables, data types, and functions. You will see and need to use loops and conditional statements throughout your coding journey; they are fundamental concepts in Python.

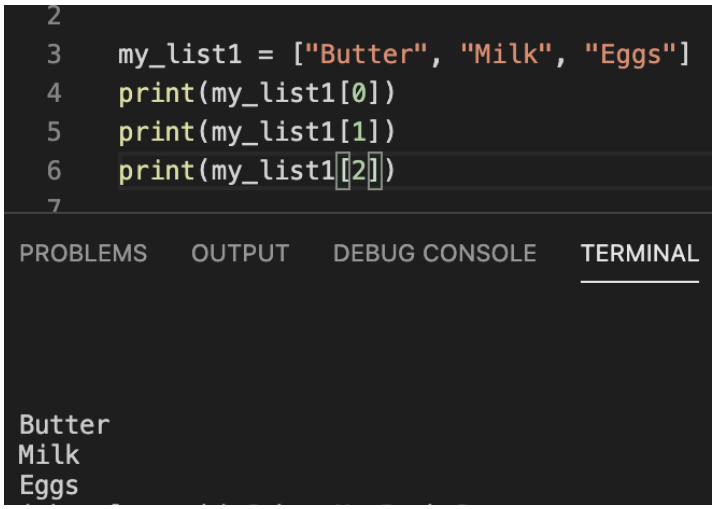
Loops

For Loop

Imagine, that we have a **list** of elements that we want to print to the terminal.



Now let's say we want to **print** each element in the **list** out one at a time, without the quotes and brackets. We could try something like this:



This solution technically works, but there is an easier, more elegant solution to this type of problem: a **for** loop. A **for** loop iterates over a sequence and performs a piece of code for all of some of the elements of that sequence. Here is an example:

```
3   my_list1 = ["Butter", "Milk", "Eggs"]
4
5   for grocery in my_list1:
6       print(grocery)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Butter
Milk
Eggs
```

Let's break this example down.

- At the top of this code snippet, we have our **my_list1** **list** from before, featuring three **string** elements.
- By typing **for**, we tell Python that we are about to configure a **for** loop.
- The word "grocery" is a user-defined, temporary variable assigned to this **for** loop. We can name this temporary variable anything we want outside of Python's already-defined words.
- I named the temporary variable "grocery", but we can call it "i", "Chewbacca", "stuff", etc.. However, we couldn't use a word such as "None" to name the temporary variable since "None" means something specific to Python already.
- The word "in" tells the **for** loop what sequence to iterate over. In our case, we want to iterate over the elements in **my_list1**.
- This initial line in a **for** loop must end in a colon, as seen after **my_list1** in our example.
- The next line in the **for** loop is tabbed. This line performs some action during each iteration of the **for** loop.
- In our case, the **for** loop says "**for** each element in **my_list1**, **print** the element to the terminal." Notice, we are not using the **print()** function on **my_list1** directly, but rather on the temporary variable, "grocery."
- Since there are three elements in **my_list1**, the **for** loop runs three times—iterating on each element one at a time—and **prints** each element to the terminal, as seen in the picture above.

While Loop

Unlike **for** loops, which execute code a number of times depending on the sequence it is iterating over, a **while** loop executes a piece of code as long as a user-defined condition is evaluated as True. If the condition becomes False, the **while** loop will stop running. Let's take a look at an example:

```
2   my_num1 = 8
3
4   while my_num1 <= 10:
5       print(my_num1, "is less than or equal to 10")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
8 is less than or equal to 10
```

In the example above:

- **my_num1** is equal to the **integer** 8.
- We then use a **while** loop to **print** **my_num1** and a **string**, as long as **my_num1** is less than or equal to the **integer** 10.
- Since 8 is less than 10, the **while** loop is True and executes…an infinite number of times; 8 remains less than 10, so the **while** loop continues to run.
- This type of behavior can incapacitate a script, computer, or server, so it is best to use **while** loops carefully.

We can use an **else** statement to tell the **while** loop to take some sort of action if the loop becomes False. Let's take a look at an example:

```
2   my_num1 = 8
3
4   while my_num1 <= 10:
5       print(my_num1, "is less than or equal to 10")
6       my_num1 = my_num1 + 1
7   else:
8       print(my_num1, "is greater than 10")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

8 is less than or equal to 10
9 is less than or equal to 10
10 is less than or equal to 10
11 is greater than 10
```

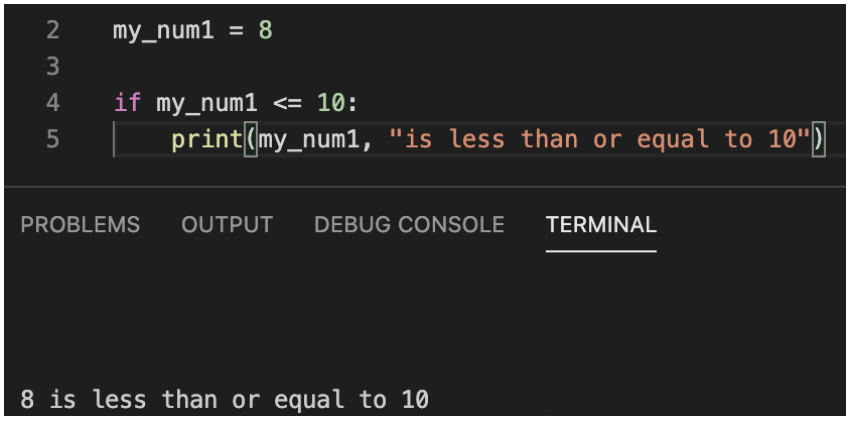
Let's break this example down:

- We see again that my_num1 is equal to the integer 8, and the while loop runs as long as my_num1 is less than or equal to the integer 10.
- Below the first print() function, we add the integer 1 to my_num1.
- What does this do? As the while loop continues to run, 1 is continuously added to my_num1. We can see the result of this code in the terminal output.
- Since adding 1 to 8 will eventually equal 11—breaking the while loop—we have an else statement that activates once the while loop becomes False, once my_num1 equals something greater than 10.
- The else statement prints the value of my_num1 along with a string, as seen in the output. This is one way to safely use a while loop, and take action when the condition becomes False.

Conditional Statements

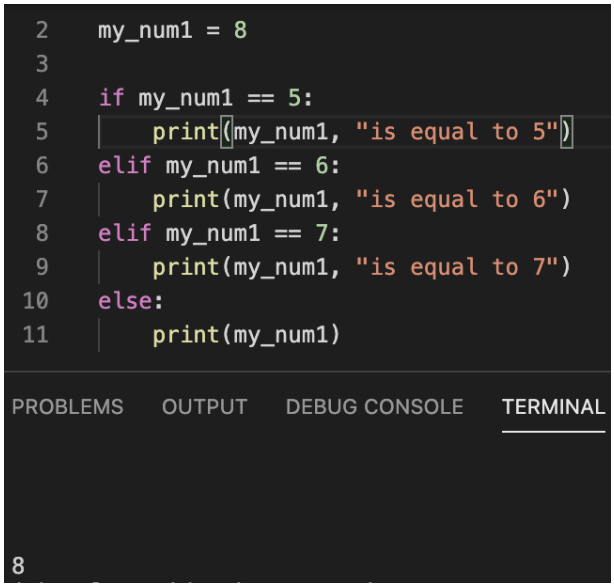
If/elif/else

Like a while loop, an if condition evaluates an expression and runs a piece of code accordingly. Let's take a look at an example:



- Like before, we define my_num1 and set it equal to the integer 8.
- This time, we use an if condition to say, "if my_num1 is less than or equal to 10… print my_num1 and a string to the terminal."
- However, unlike a while loop, the if condition runs only once; even though 8 remains less than 10, the if condition stops running after one round.

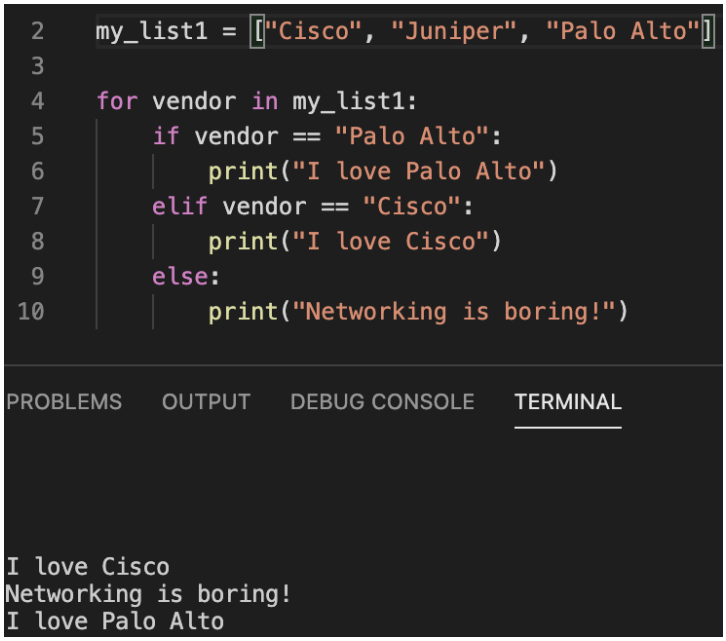
What if, however, the if condition is not True? In other words, what if my_num1 is greater than 10? We can use elif and else conditions to handle these cases. Let's take a look at an example:



- As before, we set **my_num1** equal to the **integer** 8.
- This time, we have an **if** condition saying, "**if my_num1** is equal to 5, **print** the variable and a **string**."
- Since 5 is not equal to 8, the **if** condition moves on to the first **elif** condition.
- **Elif** is a portmanteau of else and if, and will run when the **if** condition is not True.
- We have two **elif** conditions. Since the first one is not True, the second **elif** condition is evaluated. It too is not True.
- As a result, our code hits the **else** condition, which executes as a last resort if all other conditions above are not True.
- Since 8 is not equal to 5, 6, nor 7, the **else** condition is activated, **printing** 8 to the terminal output.

Nesting

Often times, you will find yourself needing to use loops and conditionals in tandem to achieve a certain goal with your code. Let's take a look at a nested example:



Let's break this down:

- At the top of the code snippet, we see a **list** entitled **my_list1**, which is comprised of three **string** elements.
- We then execute a **for** loop against this **list**, using "vendor" as the self-defined, temporary variable. This means the **for** loop will run three times, once for each element in the **list**.
- However, within the **for** loop is an **if**/**elif**/**else** conditional statement set.
- This conditional set evaluates each element of the **list** as the for loop ingests it; "Cisco" is first ingested into the **for** loop, and is ran against the **if**/**elif**/**else** conditions.
- Since "Cisco" matches the **elif** condition, the **elif** condition is executed, as seen in the terminal output.
- Next, "Juniper" is ingested into the **for** loop. But since "Juniper" does not satisfy the **if** and **elif** conditions, the **else** condition is executed.
- Lastly, "Palo Alto" is ingested in the **for** loop. Since the **if** condition matches "Palo Alto", the **if** condition is executed.

Break and Continue

Break

While looping in Python, you might want to prematurely **break** a loop when a certain condition is met. Let's take a look at an example:

```
3   my_range = range(10)
4
5   for number in my_range:
6       if number == 5:
7           break
8       print(number)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

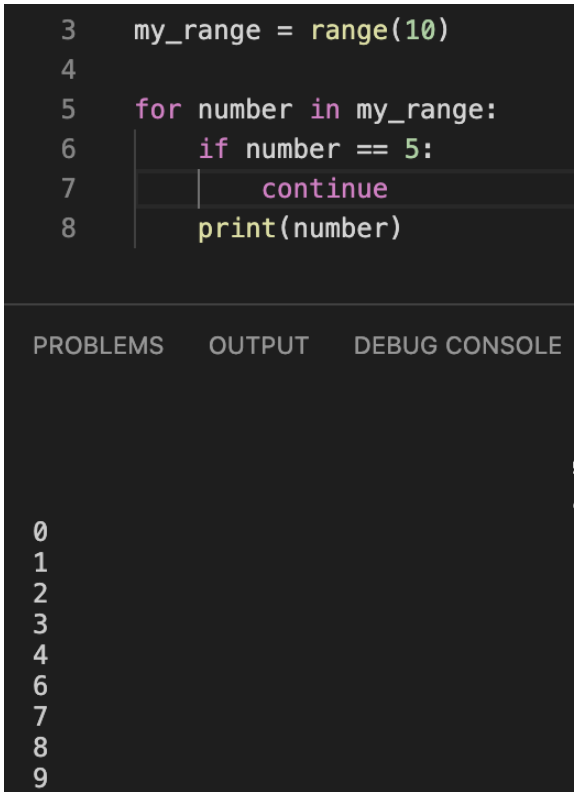
```
0
1
2
3
4
```

In the example above:

- Using the range function, we define a range of 10 **integers** via **my_range**.
- We then define an **if** condition inside of a **for** loop.
- The **for** loop will iterate over each **integer** in **my_range**.
- The **for** loop, without a **break** statement, *would* run 10 times. However, the **break** statement within the **if** condition **breaks** the loop if the **integer** ingested into the loop is equal to 5.
- If the **if** condition is not met, the **for** loop will **print** the ingested **integer** to the terminal output.
- We can see that the **for** loop only **prints** 0, 1, 2, 3, and 4 before being broken.

Continue

What if, however, we do not want the entire loop to break when an **if** condition is met, but instead just want the loop to skip an element within a list, dictionary, etc.? In this scenario, we would want to use a **continue** statement. Here's an example:



- This example looks just like the previous, but instead of a **break** statement we are using a **continue** statement.
- With the **continue** statement within the **if** condition, the **for** loop returns to the top of the loop and grabs the next **integer** in the **range** sequence when the ingested **integer** is equal to 5.
- We can confirm the **continue** statement worked by looking at the terminal output, which is missing the **integer** 5.

Conclusion

Like variables, primary data types, and functions, understanding loops and conditional statements is essential if you want to be a Python developer. Powerful while used alone, and commonly used together, loops and conditional statements will be seen through each of these lessons. The .py file **Lesson2-loops_and_conditional_statements** has examples. Use these examples as a springboard to experiment on your own before moving on to the next lesson.

Here is some further reading about loops and conditional statements:

- For Loops:
  - The basics
  - More basics, including the break and continue statements
- While Loops:
  - The basics

- If Conditional Statements:
    - The basics, with plenty of examples