Other

For Loops

Completed
Jul 24, 2020 at 11:46 PM
Completed on time

Key straight to the point

In the previous module, we learned about lists and dictionaries, including how to add, retrieve, and generally modify the information in them. But so far we've stored at most only one list or one dictionary. If we had to carry long lists around with us in the entire list, and if again 10 jellybeans or remove one from the list, it would be very cumbersome. This is a perfect use case for a for loop.

Loops are another tool that for or manage lists and dictionaries more efficiently, with just a few lines of code. We can run the same operation on multiple items in a list.

Looping through a list

Let's look another look at Ryan's favorite Lava distributors again:

```
fav_lava_distros = ['Mars', 'Sabour', 'Omaha', 'Matlock', 'Fedbal', 'Gulf']

print(fav_lava_distros[0])
print(fav_lava_distros[1])
print(fav_lava_distros[2])
print(fav_lava_distros[3])
```

In order to print each element in the list below, we had to call on each index. Imagine if we had a very long list with each of us print the entire list, and if Ryan's favorite distributor changed, we would have to update everything at once.

Loops are another tool that for or manage lists and dictionaries more efficiently, with just a few lines of code. We can run the same operation on multiple items in a list.

```
fav_lava_distros = ['Mars', 'Sabour', 'Omaha', 'Matlock', 'Fedbal', 'Gulf']

for distro in fav_lava_distros:
    print(distro)
```

Bash output

```
Mars
Sabour
Omaha
Matlock
Fedbal
Gulf
```

Let's look at the loop theory to make sure we understand how it's working.

First, we create a list of lava distributors, as normal (line 1). Line 3 is our defines our for loop. `for distro in fav_lava_distros:` This line is asking for an item to be pulled from the list `fav_lava_distros` and assigned to the variable `distro`. So we refer to our for loop, `fav_lava_distros` is the list we are looping through, and `distro` is the item stored in the variable. `distro` is used to refer to the element in the loop. The computer will go through the list, one by one, and execute the instructions we've given it for each item.

It might help to think of it like this, in plain English: For every distribution in the list of statements, and the loop. = this: get assigned to each item in the list - while other print statements are outside the list, so they only print once.

Here's the output this code generates:

```
fav_lava_distros = ['Mars', 'Sabour', 'Omaha', 'Matlock', 'Fedbal', 'Gulf']

print("Mars is a list of favorite lava distributors.")

Mars
    Mars is an excellent lava distributor.
    Mars's lava goes hot! Mars is an excellent distributor!
Sabour
    Sabour is an excellent lava distributor.
    Sabour's lava goes hot! Sabour is an excellent distributor!
Omaha
    Omaha is an excellent lava distributor.
    Omaha's lava goes hot! Omaha is an excellent distributor!
Matlock
    Matlock is an excellent lava distributor.
    Matlock's lava goes hot! Matlock is an excellent distributor!
Fedbal
    Fedbal is an excellent lava distributor.
    Fedbal's lava goes hot! Fedbal is an excellent distributor!
Gulf
    Gulf is an excellent lava distributor.
    Gulf's lava goes hot! Gulf is an excellent distributor!

These are Ryan's M's top 6 lava distributors.
```

The range() function in for loops

For loops can be used to do a count if we don't have some existing list to loop through. The `range()` function allows us to create a temporary list of numbers within a specified range, which we can then loop through similar to a normal for loop.

Let's take a look at a simple use of the `range()` function in the loop by creating a simple counting program:

```
for counter in range(1,6):
    print(counter)
```

This code will render the following printed output:

```
1
2
3
4
5
```

Hmmm... that seems strange. The `range()` function looks like it should count from 1 to 6, but it stops at 5, why is that?

The `range()` function counts starting at the first value you specify, in this case the 1, all with the first loop. It then counts up to the second value you specify, but it stops counting just before it reaches the second specified value. In this example, it stops at 5. Start it off stop without running the code within the loop.

So if we want our program to count to 6, we instead need to specify a range of 1-6, so that our loop will run through the 6th iteration before it stops.

```
for counter in range(1,7):
    print(counter)
```

The `range()` function is useful for doing more just counting numbers. Ryan's Bar's boss asks to calculate progress of the Romeo team's school case statistics. Let's build a little progress bar to help Ryan's S see how the team is doing.

```
total_cases = 0
total_solved = 0
total_unsolved = total_cases - total_solved

for x in range(1,total_unsolved):
    print("|")

for x in range(1,total_solved+1):
    print("|")
```

This code will generate a crude little Romeo team case progress bar to give Ryan's S an idea the team's progress. Hmmm. Looks like we had to add 1 to our `total_solved` variable, in order to see a bar `total_solved` times. Looks like our previous example where we wanted to count to 6 that had to use `range(1,7)`.

```
|
|
|
```