11.4.1 Use a JavaScript for Loop

The array containing UFO sightings is huge. Dana knows that iterating through it is inevitable, which means she will definitely need `for` loops. Feeling bolstered by her practice with arrow functions, Dana is ready to up the difficulty a bit by exploring how to create `for` loops in JavaScript.

All coding and scripting languages have a way to iterate through items, such as names in a list. In JavaScript, this process is initiated by the keyword "for" and works in the same manner as a Python `for` loop. Let's see how this works.

Add the following array to your console.

```
let friends = ["Sarah", "Greg", "Cindy", "Jeff"];
```

IMPORTANT

> Like Python, JavaScript uses zero-based indexing. This means that the first item in an array will be assigned an index placement of 0.

As soon as you press Enter, the words "undefined" will appear directly below your line of code. This is how you know that you've successfully executed the line of code and the array has been saved locally.

NOTE

> Code executed through the console is saved locally, within your system's memory. If you close your console and reopen it, the code will have been erased and you'll need to start over.

To iterate through each name in JavaScript, we can create a `for` loop. First, type the following in your console:

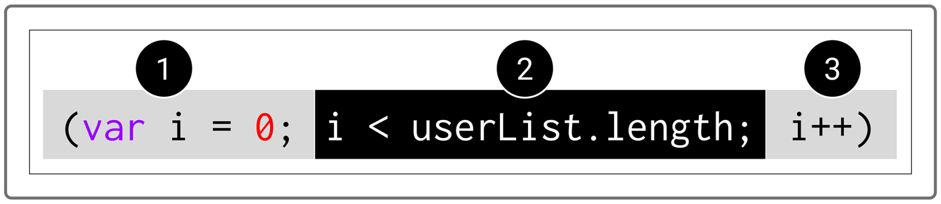
```
function listLoop(userList) {
   for (var i = 0; i < userList.length; i++) {
     console.log(userList[i]);
   }
}
```

Wow, this `for` loop is pretty involved! Let's compare it to a Python loop and examine the differences.

JavaScript `for` loop	Python `for` loop
`for (var i = 0; i < userList.length; i++) { console.log(userList[i]); }`	`for i in user_list: print(i)`

These two loops do the same thing: they iterate through a list and then print each item within it individually. The `for` keyword at the beginning of the loop is the biggest similarity here, so let's start there.

The keyword `for` is the trigger that indicates we'll be iterating through a list. The next line in the JavaScript loop does a few different things, though. This one line can be broken down into three sections.



```
(var i = 0; i < userList.length; i++)
```

The following actions occur in this one line:

1. `var i = 0` We assign an iterable variable and set its value to zero. In this loop, think of the letter 'i' to mean 'iterate.' When we assign a zero

value, we're starting a counter from the beginning. You can also think of it in terms of list comprehension–the first name of the list has an index value of zero, for example.

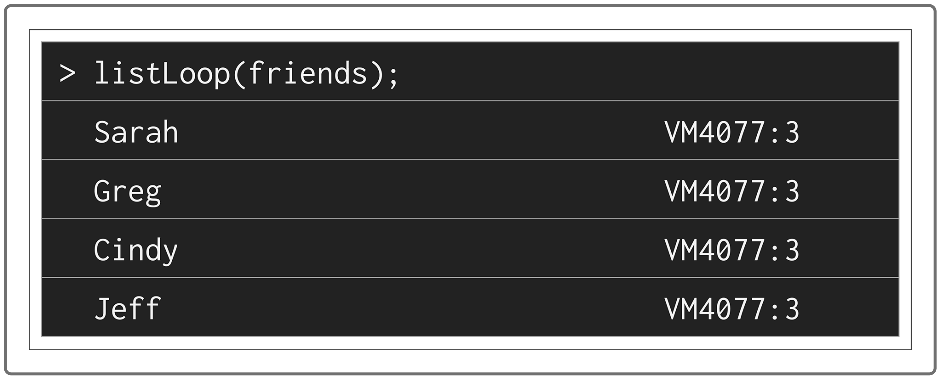
2. `i < userList.length`; Here we're basically saying, "If this iterable (`i`) is still smaller than the total number of iterables in the list (`userList`), then move on to the next step."

   So if we're on the second name, but the list is four names long, the `for` loop will continue to loop through it.

3. The final step, `i++`, increases the iterable by 1. We're using list comprehension here; the for loop knows to iterate to the next name because the index number has increased by 1.

When the length of `i` is equal to the total number of items in the list, the `for` loop will complete its iterations and the next line of code will be executed. For example, Jeff's index position is 3; when `i` is equal to 3, the loop is complete. This is because there are no names after Jeff's, nothing with an index value of 4.

Since our code says to log, or print, each iteration, the names in the array are printed to the console one at a time.



```
> listLoop(friends);
  Sarah                                  VM4077:3
  Greg                                   VM4077:3
  Cindy                                  VM4077:3
  Jeff                                   VM4077:3
```