Loops are one of the most important programming structures in the Bash language. All of the programs we've written so far are executed from the first line of the script until the last line, but loops allow you to repeat lines of code based on logical conditions or by following a sequence. The first kind of loop that we'll discuss is a FOR loop. **FOR loops** iterate through every element of a sequence that you specify. Let's take a look at a small example FOR loop:

```
#!/usr/bin/env bash
# File: forloop.sh

echo "Before Loop"

for i in {1..3}
do
    echo "i is equal to $i"
done

echo "After Loop"

```

Now let's execute this script:

```
bash forloop.sh

## Before Loop
## i is equal to 1
## i is equal to 2
## i is equal to 3
## After Loop

```

Let's walk through forloop.sh line-by-line. First "Before Loop" is printed before the FOR loop, then the loop begins. FOR loops start with the syntax for [variable name] in [sequence] followed by do on the next line. The variable name that you define immediately after for will take on a value inside of the loop that corresponds to an element in the sequence you provide after in, starting with the first element of the sequence, followed by every subsequent element. Valid sequences include brace expansions, explicit lists of strings, arrays, and command substitutions. In this instance we're using the brace expansion {1..3} which we know expands to the string "1 2 3". The code executed in each iteration of the loop is written between do and done. In the first iteration of the loop, the variable $i contains the value 1. The string "i is equal to 1" is printed to the console. There are more elements in the brace expansion after 1, so after reaching done the first time, the program starts executing back at the do statement. The second time through the loop variable $icontains the value 2. The string "i is equal to 2" is printed to the console, then the loop goes back to the do statement since there are still elements left in the sequence. The $ivariable is now equal to 3, so "i is equal to 3" is printed to the console. There are no elements left in the sequence, so the program moves beyond the FOR loop and finally prints "After Loop". Stop for a

moment and edit this loop yourself. Try changing the brace expansion to include other sequences of numbers, letters, or words, then execute the modified code. Before you execute your modified program, write down what you think will be printed. How do the results of executing your program compare with your expectations?

Once you've experimented a little take a look at this example with several other kinds of sequence generating strategies:

```
#!/usr/bin/env bash
# File: manyloops.sh

echo "Explicit list:"

for picture in img001.jpg img002.jpg img451.jpg
do
    echo "picture is equal to $picture"
done

echo ""
echo "Array:"

stooges=(curly larry moe)

for stooge in ${stooges[*]}
do
    echo "Current stooge: $stooge"
done

echo ""
echo "Command substitution:"

for code in $(ls)
do
    echo "$code is a bash script"
done
bash manyloops.sh
## Explicit list:
## picture is equal to img001.jpg
## picture is equal to img002.jpg
## picture is equal to img451.jpg
##
## Array:
## Current stooge: curly
## Current stooge: larry
## Current stooge: moe
##
## Command substitution:
## bigmath.sh is a bash script
## condexif.sh is a bash script
## forloop.sh is a bash script
## letsread.sh is a bash script
## manyloops.sh is a bash script
## math.sh is a bash script
## nested.sh is a bash script
## simpleelif.sh is a bash script
## simpleif.sh is a bash script
## simpleifelse.sh is a bash script
## vars.sh is a bash script

```

The example above illustrates three other methods of creating sequences for FOR loops: typing out an explicit list, using an array, and getting the result of a command substitution. In each case a variable name is declared after the for, and the value of tha variable changes through each iteration of the loop until the corresponding sequence has been exhausted. Right now you should take a moment to write a few FOR loops yourself, generating sequences in all of the ways that we've gone over, just to reinforce your understanding of how a FOR loop works. Loops and conditional statements are two of the most important structures that we have at our disposal as programmers.

Mark as completed