
# Pseudo Code for Iteration

In addition to using boolean conditions to control the flow through your program,I'm going to tell you a little it about the `for` statement, and how that works. I'm going to show you some pseudo code for how you might want to iterate through an array. An `iteration`, in this case, would be repeating a block of statements. JavaScript lets you group statements together to be repeated in blocks and then use these control flow statements to execute them.

```
var colors = ["red", "green", "blue"];
var question;
var counter;

for(
  counter = 0; // initial counter state
  counter < colors.length; // terminal counter state
  counter++) // iteration for each loop
  { // block of code
    question = "Do you like " + colors[counter] + "?";
    console.log(question);
    // "Do you like red?" (first time)
    // "Do you like green?" (second time)
    // "Do you like blue?" (third time)
  }
```

# "For" Loop Iteration

A `for` loop lets you iterate, and there are four elements to a For statement.

- The first is the initial condition that defines what the state of the world is when the loop begins.
- Next is the final condition,that says, how does the loop know when to stop executing.
- The third is an iterator, which says, what changes forone loop to the next in order to know that you're advancing.
- The fourth is a block of code that you want to execute.

I'm going to show you the code for how we can accomplish what we were trying to do in our pseudocode.

```
var colors = ["red", "green", "blue"];
var question;
var counter; // Declare iterator variables first

for(counter = 0; counter < colors.length; counter++) {
  question = "Do you like " + colors[counter] + "?";
  console.log(question);
}
```

`counter` is a variable that says what index point in the array we want to be at.

`counter = 0` is the starting state.

`counter < colors.length` is the final condition when we want the loop to stop executing.

`counter++` means that on each iteration we want to increment our counter by one.

`for` loops are one of the basic tools for flow control in a JavaScript program. I just want to take you through the four elements of a `for` statement in a little bit more detail so you're more familiar with how they work.

# Initial Conditions

We'll start with the initial conditions. It's important to remember to declare any variables that you are going to be using in a loop first. And also remember not to reuse the same iterator variable from one loop to the next.I f you look at JavaScript programs written by other programmers, you may see them using the same iterator over and over again. If you're careful, you can get away with it. But you must be very careful, because sometimes you can forget that an iterator has been set to a new value before starting again. And if you don't three set the initial conditions properly, you can get into trouble.

# Terminal Conditions

The terminal conditions of a `for` loop tell the loop when it's time to stop looping. This is structured just like a boolean. As long as the value returned by the terminal condition is `true`, the loop will continue executing. Once that value returns as `false`, then the loop stops executing and continues on to whatever is past it in the code.

# Iteration of "For" Loop

The iteration part of a `for` loop tells what changes between each loop in order to move the process forward. And in the iterator, you can change anything that you like. Usually you upgrade a counter, sometimes by one, sometimes by more than one - it depends on what you're program is actually trying to accomplish.

An interesting thing to note is the actually expressions inside a `for` loop are all optional. You don't need an initial condition, a terminal condition, an iteration, o reven a block of code to execute. If you create a `for` loop that looks like this

```
for(;;) {
  console.log("this could go on forever");
  // Don't try this or your environment
  // could get trapped in an infinite loop
}
```

which doesn't have an initial condition, a terminal condition, or an iterator, you're basically creating an infinite loop. JavaScript will not stop you from creating an infinite loop - it's going to continue iterating forever and ever and ever, until you do something about it.

# The "Break" Statement

Because of that, it's useful to know about the `break` statement. It lets you break out of a loop if a certain condition is met. It will interrupt the loop before the terminal condition is reached.

```
var counter = 0; // initial condition

for(;;) { // will loop infinitely until a break
  console.log("this could go on forever");
  counter++; // iterator
  if(counter >= 10) { // terminal condition
    break;
  }
}
```

So the `break` statement tells a `for` loop that's running to go ahead and stop running even if it hasn't reached it's terminal condition.

There may be times when you're going to want to let the conditions of a program determine when it should break out of a `for` loop before the terminal condition is reached. So it's useful to know about break statements. It's unusual to see a for loop without an initial state, a terminal state, and an iterator, or even a block to execute, but it's good to know that it's possible.