Javascript Basics

Loops

Computers excel at doing repetitious tasks quickly. Most programming languages have statements for executing a loop of code

Two common loop structures in Javascript are the `while` and `for` loops. While loops are slightly simpler, so we'll start with those.

Let's write a conditional that outputs a number and increments it by 1 if the number is less than 10:

```
var number = 0
if (number < 10) {
  console.log(number)
  number = number + 1
}
console.log('Finished with number at ' + number)
```

Now let's say we want to output numbers 0 through 9. The statement under the if condition will only be called once, but if we replace `if` with `while`, it will be called until the condition isn't met anymore:

```
var number = 0
while (number < 10) {
  console.log(number)
  number = number + 1
}
console.log('Finished with number at ' + number)
```

While loops tend to be dangerous because if programmers don't write the condition correctly, the loop will never finish.

If that `number = number + 1` line wasn't there, that loop would go on forever, and would likely crash your browser tab.

We can shorten the line that increments the number a bit with the increment operator. The increment operator adds one to an integer and returns the result. Unlike the addition operator, it actually affects (mutates) the value in memory.

The increment operator returns the integer value BEFORE incrementing if the operator is placed AFTER the integer, and AFTER incrementing if placed BEFORE:

```
var x = 5
var y = x++ // y = 5 | x = 6

var x = 5
var y = ++x // y = 6 | x = 6
```

So knowing that, the while loop becomes:

```
var number = 0
while (number < 10) {
  console.log(number)
  number++
}
console.log('Finished with number at ' + number)
```

Quick aside, there's also a construct known as the do while loop in cases where you want the code in the loop condition to always execute at least once. It looks like this:

```
var number = 0
do {
  console.log(number)
  number++
} while (number < 10)
console.log('Finished with number at ' + number)
```

In this case, it actually behaves the same way, but if number was already greater than 10, `do while` would output and increment the number once.

```
var number = 20
do {
  console.log(number)
  number++
} while (number < 10)
console.log('Finished with number at ' + number)
```

Typically, when iterating through a loop we have a known starting point, a condition that specifies when we're done looping, and an amount to increment by (usually 1).

```
var number = 0 // start
while (number < 10) { // condition
  console.log(number) // statement
  number++ // increment
}
for (var number = 0; number < 10; number++) {

}
console.log('Finished with number at ' + number)
```

The for loop is a short way to combine those steps into one line. The syntax looks like this:

```
for ( start; condition; increment ) {
  // statement
}
```

Each part of the loop is separated by semicolons.

The start or initalization step is called before the loop begins, and is typically used to declare variables.

The condition is what is evaluated at each loop step, and like the condition in the while loop, once it's no longer met, the loop completes. Make sure to include a condition because the default behavior is to continue the loop indefinitely.

The increment or final-expression is executed after the statement is evaluated in each loop iteration.

Let's convert the while loop into a for loop:

```
for (var number = 0; number < 10; number++) { // start; condition; increment
  console.log(number) // statement
}
console.log('Finished with number at ' + number)
```

There shouldn't be anything mysterious going on with the loop. In fact, in this case, every part of the loop can handled outside of the parentheses. The variable initalization and increment steps can be handled like this:

```
var number = 0
for (; number < 10;) { // start; condition; increment
  console.log(number) // statement
  number++
}
console.log('Finished with number at ' + number)
```

Now that we know a bit about `for` and `while` loops, I'll introduce `break` and `continue` statements.

The `break` statement tells Javascript to exit out of a loop. Its' useful when the loop needs to check multiple conditions. For example, a loop might be used to search for a particular word in an array of strings, and once it's found, the loop doesn't need to keep going.

Using the loop from earlier, we can jump out early once the number reaches 5:

```
for (var number = 0; number < 10; number++) { // start; condition; increment
  console.log(number) // statement
  if (number === 5) {
    break
  }
}
console.log('Finished with number at ' + number)
```

Notice that when breaking out of a loop, the increment step doesn't execute. Number is still 5.

And like the `return` statement, break stops any further execution of the loop statement. So if that console log was placed under the if check, the number 5 would not be shown (until the last console message):

```
for (var number = 0; number < 10; number++) { // start; condition; increment
  if (number === 5) {
    break
  }
  console.log(number) // statement
}
console.log('Finished with number at ' + number)
```

Next, the `continue` statement. `continue` tells Javascript to *continue* through the loop and ignore any code after in the statement. If we replaced `break` with `continue` in this code, it would still run the statement for each step in the loop, but it would not output the number if it was 5:

```
for (var number = 0; number < 10; number++) { // start; condition; increment
  if (number === 5) {
    continue
  }
  console.log(number) // statement
}
console.log('Finished with number at ' + number)
```

And remember, `break` and `continue` work in both for and while loops:

```
var number = 0
while (number < 10) {
  number++ // make sure to increment BEFORE continue!
  if (number === 5) {
    continue
  }
  console.log(number - 1) // -1 because we already incremented
}
console.log('Finished with number at ' + number)
```

Now that we know how to use loops, we'll go over the Fizz-Buzz test.