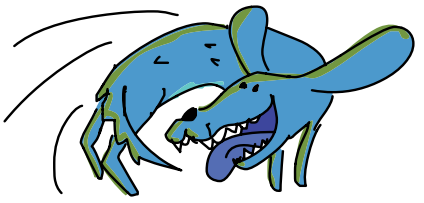
LOOPS

As we've seen, conditionals allow you to run a piece of code once if a condition is `true`. Loops, on the other hand, allow you to run a piece of code multiple times, depending on whether a condition remains `true`. For example, while there's food on your plate, you should keep eating; or, while you still have dirt on your face, you should keep washing.



WHILE LOOPS

The simplest kind of loop is a `while` loop. A `while` loop repeatedly executes its body until a particular condition stops being `true`. By writing a `while` loop, you are saying, "Keep doing this while this condition is `true`. Stop when the condition becomes `false`."

As Figure 6-4 shows, `while` loops start with the `while` keyword, followed by a condition in parentheses and then a body in braces.

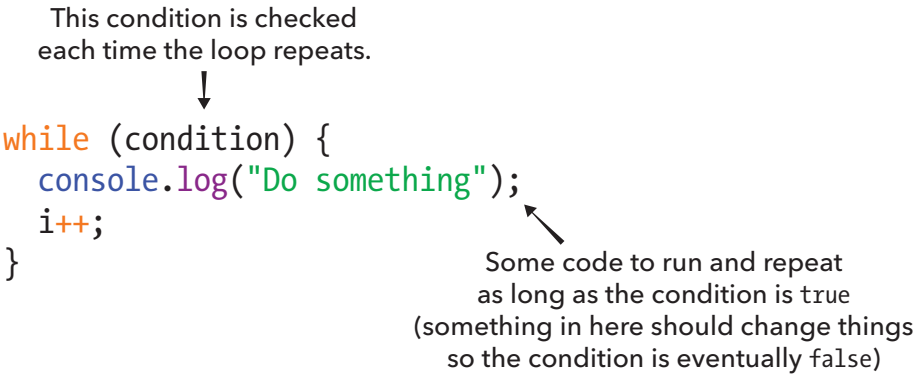


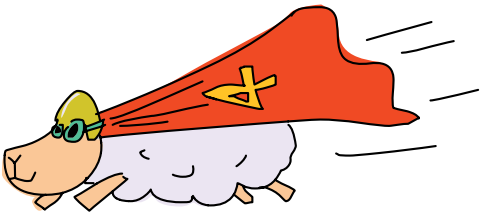
Figure 6-4: The general structure of a `while` loop

Like an `if` statement, the body of a `while` loop is executed if the condition is `true`. *Unlike* an `if` statement, after the body is executed, the condition is checked again, and if it's still `true`, the body runs again. This cycle goes on until the condition is `false`.

COUNTING SHEEP WITH A WHILE LOOP

Say you're having trouble sleeping and you want to count sheep. But you're a programmer, so why not write a program to count sheep for you?

```
var sheepCounted = 0;
❶ while (sheepCounted < 10) {
❷ console.log("I have counted " + sheepCounted + " sheep!");
  sheepCounted++;
}
console.log("Zzzzzzzzzz");
```



We create a variable called `sheepCounted` and set its value to 0. When we reach the `while` loop ❶, we check to see whether `sheepCounted` is less than 10. Because 0 is less than 10, the code inside the braces (the body of the loop) ❷ runs, and `"I have counted " + sheepCounted + " sheep!"` is logged as "I have counted 0 sheep!" Next, `sheepCounted++` adds 1 to the value of `sheepCounted`, and we go back to the start of the loop, over and over:

```
I have counted 0 sheep!
I have counted 1 sheep!
I have counted 2 sheep!
I have counted 3 sheep!
I have counted 4 sheep!
I have counted 5 sheep!
I have counted 6 sheep!
I have counted 7 sheep!
I have counted 8 sheep!
I have counted 9 sheep!
Zzzzzzzzzz
```

This repeats until `sheepCounted` becomes 10, at which point the condition becomes `false` (10 is *not* less than 10), and the program moves on to whatever comes after the loop. In this case, it prints `Zzzzzzzzzz`.

PREVENTING INFINITE LOOPS

Keep this in mind when you're using loops: if the condition you set never becomes `false`, your loop will loop forever (or at least until you quit your browser). For example, if you left out the line `sheepCounted++;`, then `sheepCounted` would remain 0, and the output would look like this:

```
I have counted 0 sheep!
I have counted 0 sheep!
I have counted 0 sheep!
I have counted 0 sheep!
...
```

Because there's nothing to stop it, the program would keep doing this forever! This is called an *infinite loop*.

FOR LOOPS

`for` loops make it easier to write loops that create a variable, loop until a condition is `true`, and update the variable at the end of each turn around the loop. When setting up a `for` loop, you create a variable, specify the condition, and say how the variable should change after each cycle—all before you reach the body of the loop. For example, here's how we could use a `for` loop to count sheep:

```
for (var sheepCounted = 0; sheepCounted < 10; sheepCounted++) {
  console.log("I have counted " + sheepCounted + " sheep!");
}
console.log("Zzzzzzzzzz");
```

As Figure 6-5 shows, there are three parts to this `for` loop, separated by semicolons: the setup, condition, and increment.

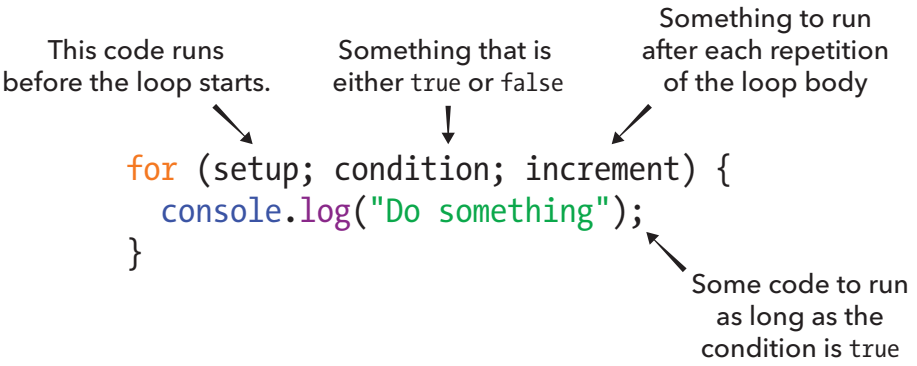


Figure 6-5: The general structure of a `for` loop

The *setup* (`var sheepCounted = 0`) is run before the loop starts. It's generally used to create a variable to track the number of times the loop has run. Here we create the variable `sheepCounted` with an initial value of `0`.

The *condition* (`sheepCounted < 10`) is checked before each run of the loop body. If the condition is `true`, the body is executed; if it's `false`, the loop stops. In this case, the loop will stop once `sheepCounted` is no longer less than 10.

The *increment* (`sheepCounted++`) is run after every execution of the loop body. It's generally used to update the looping variable. Here, we use it to add 1 to `sheepCounted` each time the loop runs.

`for` loops are often used to do something a set number of times. For example, this program will say `Hello!` three times.

```
var timesToSayHello = 3;
for (var i = 0; i < timesToSayHello; i++) {
  console.log("Hello!");
}
```

Here is the output:

```
Hello!
Hello!
Hello!
```

If we were the JavaScript interpreter running this code, we would first create a variable called `timesToSayHello` and set it to 3. When we reach the `for` loop, we run the setup, which creates a variable `i` and sets it to 0. Next, we check the condition. Because `i` is equal to 0 and `timesToSayHello` is 3, the condition is `true`, so we enter the loop body, which simply outputs the string `"Hello!"`. We then run the increment, which increases `i` to 1.

Now we check the condition again. It's still `true`, so we run the body and increment again. This happens repeatedly until `i` is equal to 3. At this point, the condition is `false` (3 is not less than 3), so we exit the loop.

USING FOR LOOPS WITH ARRAYS AND STRINGS

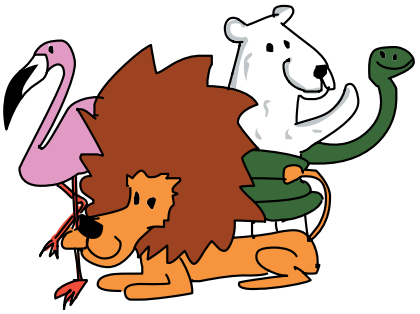
One very common use of `for` loops is to do something with every element in an array or every character in a string. For example, here is a `for` loop that prints out the animals in a zoo:

```
var animals = ["Lion", "Flamingo", "Polar Bear", "Boa Constrictor"];

for (var i = 0; i < animals.length; i++) {
  console.log("This zoo contains a " + animals[i] + ".");
}
```

In this loop, `i` starts at 0 and goes up to one less than `animals.length`, which in this case is 3. The numbers 0, 1, 2, and 3 are the indexes of the animals in the `animals` array. This

means that every time around the loop, `i` is a different index, and `animals[i]` is another animal from the `animals` array. When `i` is 0, `animals[i]` is `"Lion"`. When `i` is 1, `animals[i]` is `"Flamingo"`, and so on.



Running this would output:

```
This zoo contains a Lion.
This zoo contains a Flamingo.
This zoo contains a Polar Bear.
This zoo contains a Boa Constrictor.
```

As you saw in Chapter 2, you can access individual characters in a string in the same way you can access individual elements in an array, using square brackets. This next example uses a `for` loop to print out the characters in a name:

```
var name = "Nick";

for (var i = 0; i < name.length; i++) {
  console.log("My name contains the letter " + name[i] + ".");
}
```

This would output:

```
My name contains the letter N.
My name contains the letter i.
My name contains the letter c.
My name contains the letter k.
```

OTHER WAYS TO USE FOR LOOPS

As you might imagine, you don't always have to start the looping variable at 0 and increment it by 1. For example, here's a way to print all the powers of 2 below the number 10,000:

```
for (var x = 2; x < 10000; x = x * 2) {
  console.log(x);
}
```

We set `x` to 2 and increment the value of `x` using `x = x * 2;`, which will double the value of `x` each time the loop runs. The result gets big very quickly, as you can see:

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
```

And voilà! This short `for` loop prints out all the powers of 2 below 10,000.

TRY IT OUT!

Write a loop to print the powers of 3 under 10,000 (it should print 3, 9, 27, etc.).

Rewrite this loop with a `while` loop. (Hint: Provide the setup *before* the loop.)

WHAT YOU LEARNED

In this chapter, you learned about conditionals and loops. Conditionals are used to run code only when a certain condition is `true`. Loops are used to run code multiple times and to keep running that code as long as a certain condition is `true`. You can use conditionals to make sure that the right code is run at the right time, and you can use loops to keep your program running as long as necessary. Having the ability to do these two things opens up a whole new world of programming possibilities.

In the next chapter, we'll use the power of conditionals and loops to make our first real game!