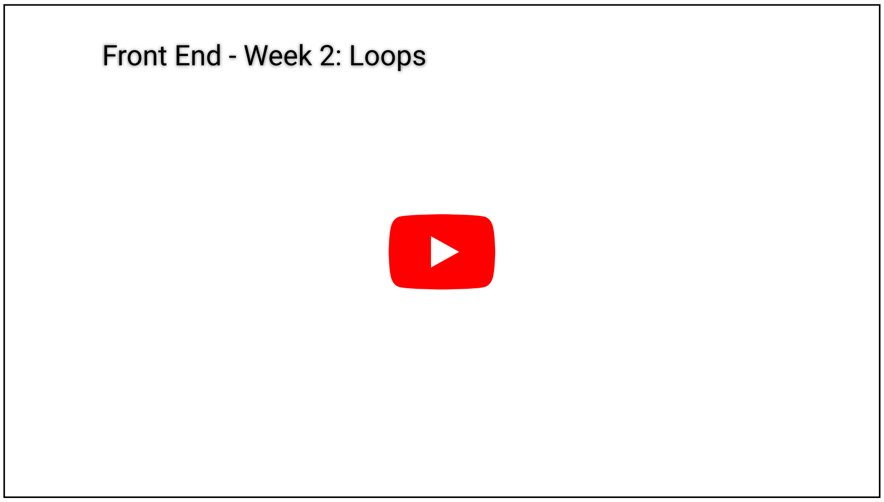
Week 2: Weekly Videos and Curriculum

2. Loops

Loops



Let's say we need to do something over and over again until some condition is met. For example, if we are baking a cake and a recipe calls for 5 cups of flour, we need to scoop a cup of flower and put it into our mixing bowl over and over until the bowl has 5 cups of flour. A simple example of this in JavaScript could look something like this:

```
var cupsOfFlour = 0;

console.log('Scooping a cup of flour into the bowl');

cupsOfFlour += 1;

console.log('There are ' + cupsOfFlour + ' cups of flour in the bowl');

console.log('Scooping a cup of flour into the bowl');

cupsOfFlour += 1;

console.log('There are ' + cupsOfFlour + ' cups of flour in the bowl');

console.log('Scooping a cup of flour into the bowl');

cupsOfFlour += 1;

console.log('There are ' + cupsOfFlour + ' cups of flour in the bowl');

console.log('Scooping a cup of flour into the bowl');

cupsOfFlour += 1;

console.log('There are ' + cupsOfFlour + ' cups of flour in the bowl');

console.log('Scooping a cup of flour into the bowl');

cupsOfFlour += 1;

console.log('There are ' + cupsOfFlour + ' cups of flour in the bowl');
```

There are two problems with the code above. While it does exactly what we're looking for, it

1) contains duplicate code and

2) it is not dynamic.

For the **first** problem, it contains duplicate code - as developers, we never want duplicate code. Duplicated code means more places to make changes if we ever need to change anything, and more places to make changes means more opportunities for something to be missed or a mistake to be made.

> We want to follow the **DRY** principle - **Don't Repeat Yourself**.

For the **second** problem, what if the requirement changed to 6 or 7 cups? What if it changed to 4? Both situations would cause us to need to change the code. This is a brittle solution that breaks as soon as the number of cups of flour required changes. Which leads to the problem, what do we do when we need code that will repeat until a condition is met?

Enter, **Loops**.

Loops do just that. Similar to an `if` statement, loops contain parentheses with specific conditions and a body denoted by curly braces that will execute again and again until the condition in the parentheses evaluates to false. There are many different kinds of loops.

Let's take a look at our first loop, the `while` Loop:

```
var cupsOfFlour = 0;

while (cupsOfFlour < 5) {

  console.log('Scooping a cup of flour into the bowl');

  cupsOfFlour += 1;

  console.log('There are ' + cupsOfFlour + ' cups of flour in the bowl');

}
```

The above solution is a much cleaner way to do the exact same thing as the 16 lines in our first example and it solves both problems.

1) it does not contain duplicate code; it is DRY.

2) we can change the condition to 6, 7, 4, or 10000000 by simply switching the number in the Boolean expression.

The way this works is that the Boolean expression `cupsOfFlour < 5` will evaluate and if it's `true` the code inside the body (between the opening and closing curly brace) will execute. After it executes, the Boolean expression will then evaluate again. If it is `true`, the body will again execute. Each execution is known as an iteration. The loop will continue to iterate until the Boolean expression evaluates to `false`. This is why the line `cupsOfFlour += 1;` is so important, because without it, cupsOfFlour will remain 0 and the Boolean expression will never be `false`, thus resulting in an infinite loop (a loop that never ends).

`for` Loop:

The next type of loop is a `for` Loop. `for` Loops contain a bit more syntax, but do more with fewer lines. Let's look at the same example as above converted to a `for` Loop:

```
for (var cupsOfFlour = 0; cupsOfFlour < 5; cupsOfFlour++) {
  console.log('Scooping a cup of flour into the bowl');
  console.log('There are ' + cupsOfFlour + ' cups of flour in the bowl');
}
```

Notice this loop is two lines shorter and does the exact same thing. A `for` Loop has three sections inside its parentheses separated by two semicolons.

- The first section is where we can declare any variables to be used in the loop. In this situation, we set our `cupsOfFlour` variable (the most common variable here in a `for` loop is `i` - read on for an example).

- The second section is where we put our Boolean expression, in this case `cupsOfFlour < 5` that determines whether or not the loop performs an iteration.

- The final section is the post-iteration, it is what happens after the loop completes an iteration. In this case, we use `cupsOfFlour++` here, which is essentially the same thing as `cupsOfFlour += 1`.

Another `for` Loop Example:

As stated above, `i` is the most common variable name in a `for` Loop. Here is an example that prints from 0 to 9:

```
for (var i = 0; i < 10; i++) {
  console.log(i);
}
```

We can also use loops and conditionals together, they don't have to be separate. We can have loops inside of `if` statements, or vice versa. Here is an example of an `if` statement inside of a loop that prints out every number from 0 to 99 divisible by 3.

```
for (var i = 0; i < 100; i++) {
  if (i % 3 == 0) {
    console.log(i);
  }
}
```

`do while` Loop:

Another type of loop is called a `do while` loop. This loop functions much like a `while` loop, except that a `while` loop has the possibility of never running if its Boolean expression evaluates to `false` the first time, and a `do while` loop will always execute at least once since the expression is at the end.

Let's take a look at an example:

```
let i = 10;

do {
  i++;
  console.log(i);
} while (i < 3);
```

As we can see here, `i` is already greater than 3, but this loop will still iterate once and then exit.