Intro to Javascript

Conditionals

DIGRESSION: GEORGE BOOLE

This happy man is George Boole (1815–1864), a 19th-century mathematician who felt that he understood how the brain worked: it calculated "truth" by performing a kind of "algebra" on a bunch of truth claims.

Statements were either true or false and the brain either AND'd them, or OR'd them, or negated them with NOT. At the end, the brain concluded whether the combined claims at hand amounted to either True or False.

That's not how the brain works, but Boole's algebra was resurrected when binary computers needed a way to calculate the truthhood or falsehood of statements. Every computer language today has a boolean data type.

Javascript, too, knows true and false.

RANDOM NUMBERS

Let's flip a coin. Since Javascript doesn't know coins or how to flip them, we'll simulate the process by choosing a random number. `Math.random()` chooses a fractional number between 0 and 1, which we can see in the console window:

```
> Math.random()
  0.9966547114435647
> Math.random()
  0.5266016283850623
> Math.random()
  0.5465462678199808
> Math.random()
  0.5151639122243392
```

We'll say that a number less than 1/2 is Heads and a number more than 1/2 is Tails.

IF / ELSE

So we'll create a <script> container and put some Javascript into it. First, we'll choose our random number:

```
<script>
   coin = Math.random();
   if (coin < 0.5) {
      document.writeln("It's heads!");
   } else {
      document.writeln("It's tails!");
   }
</script>
```

IF / ELSE

And then we'll have Javascript decide what to do based on the number we've chosen. This structure is known as a *conditional* or an *if/else* statement:

```
<script>
    coin = Math.random();
    if (coin < 0.5) {
        document.writeln("It's heads!");
    } else {
        document.writeln("It's tails!");
    }
</script>
```

IF / ELSE

In Javascript, we often need to take different actions depending on circumstances. We can use a *conditional statement* to have Javascript make decisions:

```
<script>
   coin = Math.random();
   if (coin < 0.5) {
      document.writeln("It's heads!");
   } else {
      document.writeln("It's tails!");
   }
</script>
```

This part is a boolean test. It evaluates to either true or false.

IF / ELSE

In Javascript, we often need to take different actions depending on circumstances. We can use a *conditional statement* to have Javascript make decisions:

```
<script>
   coin = Math.random();
   if (coin < 0.5) {
      document.writeln("It's heads!");
   } else {
      document.writeln("It's tails!");
   }
</script>
```

If it's true, then this block gets executed. Note the curly braces!

IF / ELSE

In Javascript, we often need to take different actions depending on circumstances. We can use a *conditional statement* to have Javascript make decisions:

```
<script>
   coin = Math.random();
   if (coin < 0.5) {
      document.writeln("It's heads!");
   } else {
      document.writeln("It's tails!");
   }
</script>
```

If it's false, then the first block gets skipped and this one gets executed.

= VS ==

=

In Javascript and other languages, = is a verb that says, “figure out what’s on the righthand side, no matter how complex, and move it to the variable on the lefthand side.”

==

But == is a boolean test that says, “True or False: do these two things have the same value?”

Javascript also uses === to test for equal value AND same data type.

IF / ELSE

Here's an example of Javascript behaving differently depending on the value in the variable name:

```
name = 'Goofus';
if (name == 'William') {
    document.write("Do people call you Bill?");
} else {
    document.write("Hello, " + name);
}
```

IF

We don't always need to have an else clause. Because this conditional will evaluate to false, nothing happens:

```
name = 'Goofus';
if (name == 'William') {
    document.write("Do people call you Bill?");
}
```

BOOLEAN OPERATORS

Javascript has a whole range of boolean operators:

SYMBOL	MEANING
==	is equal
!	not
!=	is not equal
&&	and
||	or
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
in	key of an array or an object
===	are the same object

Intro to Javascript

Loops

while

INTRO TO LOOPS: COUNT TO THREE

In Javascript, we will often want to repeat a process until some condition is achieved or until some change is complete. The programming structure we use to accomplish that is called a *loop*.

Let's think about this structure in English first:

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. Say our count out loud.
    (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
    a. If not, go to Step 2.
    b. If yes, stop.

Even if it's not obvious, there are three interesting ideas built into this model:

INTRO TO LOOPS: COUNT TO THREE

In Javascript, we will often want to repeat a process until some condition is achieved or until some change is complete. The programming structure we use to accomplish that is called a *loop*.

Let's think about this structure in English first:

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. Say our count out loud.
    (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
    a. If not, go to Step 2.
    b. If yes, stop.

Even if it's not obvious, there are three interesting ideas built into this model:

- An initial status or condition. ("we're at zero.")
- A boolean (true/false) test of some kind. ("Are we at 3 yet?")
- A degree of change. ("Add 1 to our count").

INTRO TO LOOPS: COUNT TO THREE

In Javascript, we will often want to repeat a process until some condition is achieved or until some change is complete. The programming structure we use to accomplish that is called a *loop*.

Let's think about this structure in English first:

1. **To begin, we haven't counted anything. We're at zero.**
2. Add 1 to our count.
3. Say our count out loud.
    (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
    a. If not, go to Step 2.
    b. If yes, stop.

Even if it's not obvious, there are three interesting ideas built into this model:

- An initial status or condition. ("we're at zero.")

INTRO TO LOOPS: COUNT TO THREE

In Javascript, we will often want to repeat a process until some condition is achieved or until some change is complete. The programming structure we use to accomplish that is called a *loop*.

Let's think about this structure in English first:

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. Say our count out loud.
    (In Javascript, we'll probably document.write() it.)
4. **Are we at 3 yet?**
    a. If not, go to Step 2.
    b. If yes, stop.

Even if it's not obvious, there are three interesting ideas built into this model:

- An initial status or condition. ("we're at zero.")
- A boolean (true/false) test of some kind. ("Are we at 3 yet?")

INTRO TO LOOPS: COUNT TO THREE

In Javascript, we will often want to repeat a process until some condition is achieved or until some change is complete. The programming structure we use to accomplish that is called a *loop*.

Let's think about this structure in English first:

1. To begin, we haven't counted anything. We're at zero.
2. **Add 1 to our count.**
3. Say our count out loud.
    (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
    a. If not, go to Step 2.
    b. If yes, stop.

Even if it's not obvious, there are three interesting ideas built into this model:

- An initial status or condition. ("we're at zero.")
- A boolean (true/false) test of some kind. ("Are we at 3 yet?")
- A degree of change. ("Add 1 to our count").

THREE PARTS OF A LOOP

Every computer loop has these three parts, even if they seem somewhat "buried" or "implicit." By changing one or more of the three fundamental parts, we can control how long the loop repeats its work.

1. An initial status or condition.

   This establishes the status of the universe or the program or the simulation before the loop begins.

2. A boolean (true/false) test of some kind.

   This controls when the loop finishes.

3. A degree of change.

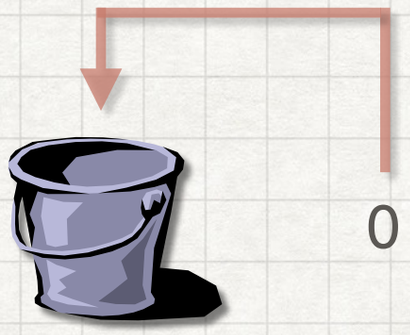
   Every iteration through the loop will make an incremental change to this feature.

A WHILE LOOP IN JAVASCRIPT

Let's slowly convert our English procedure into Javascript. Our initial condition is some data and we already suspect we'll have to put that data into a container:

1. **To begin, we haven't counted anything. We're at zero.**
2. Add 1 to our count.
3. Say our count out loud.
   (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
   a. If not, go to Step 2.
   b. If yes, stop.

```
count = 0;
```

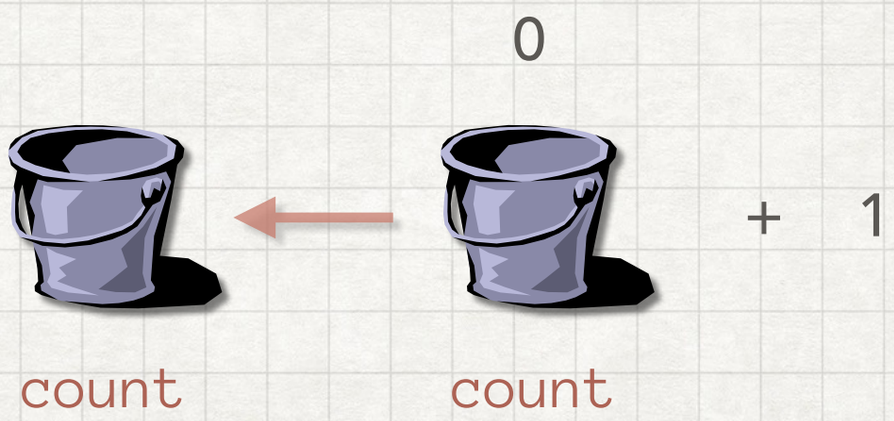


A WHILE LOOP IN JAVASCRIPT

We add one to our count. More precisely, we *increment the value in* `count`.

1. To begin, we haven't counted anything. We're at zero.
2. **Add 1 to our count.**
3. Say our count out loud.
   (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
   a. If not, go to Step 2.
   b. If yes, stop.

```
count = count + 1;
```



A WHILE LOOP IN JAVASCRIPT

We output the value in count somehow: probably document.write() it.

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. **Say our count out loud.**
   (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
   a. If not, go to Step 2.
   b. If yes, stop.

```
document.write(count);
```

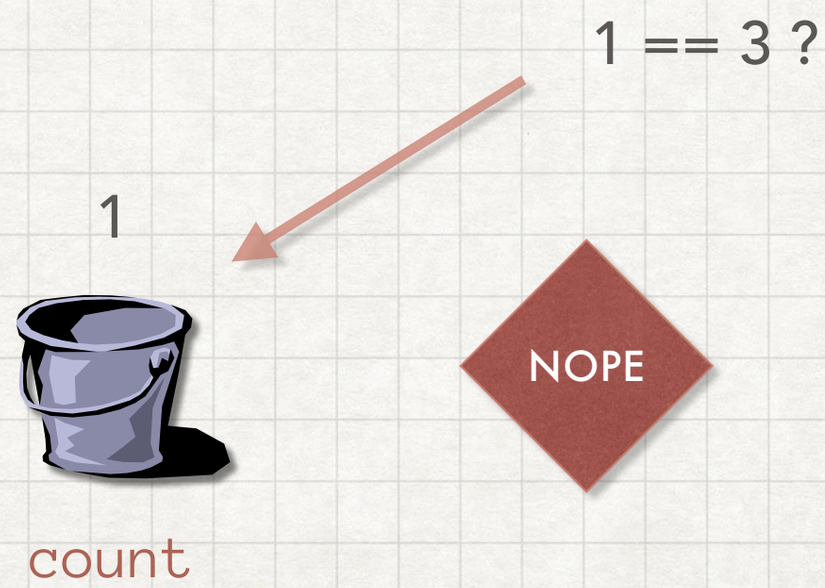
count

A WHILE LOOP IN JAVASCRIPT

Test the value in count. Is it three?

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. Say our count out loud.
   (In Javascript, we'll probably document.write() it.)
4. **Are we at 3 yet?**
   a. **If not, go to Step 2.**
   b. If yes, stop.

count == 3 ??

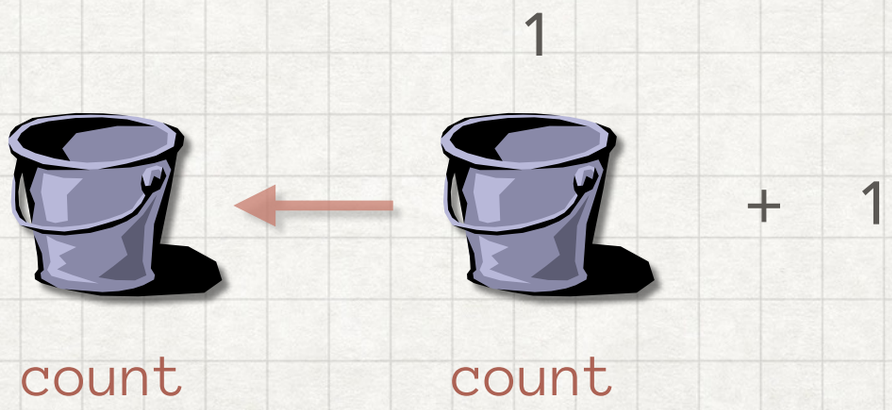


A WHILE LOOP IN JAVASCRIPT

We add one to our count. More precisely, we *increment the value in* `count`.

1. To begin, we haven't counted anything. We're at zero.
2. **Add 1 to our count.**
3. Say our count out loud.
   (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
   a. If not, go to Step 2.
   b. If yes, stop.

```
count = count + 1;
```



A WHILE LOOP IN JAVASCRIPT

We output the value in count somehow: probably document.write() it.

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. **Say our count out loud.**
   (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
   a. If not, go to Step 2.
   b. If yes, stop.

```
document.write(count);
```

2

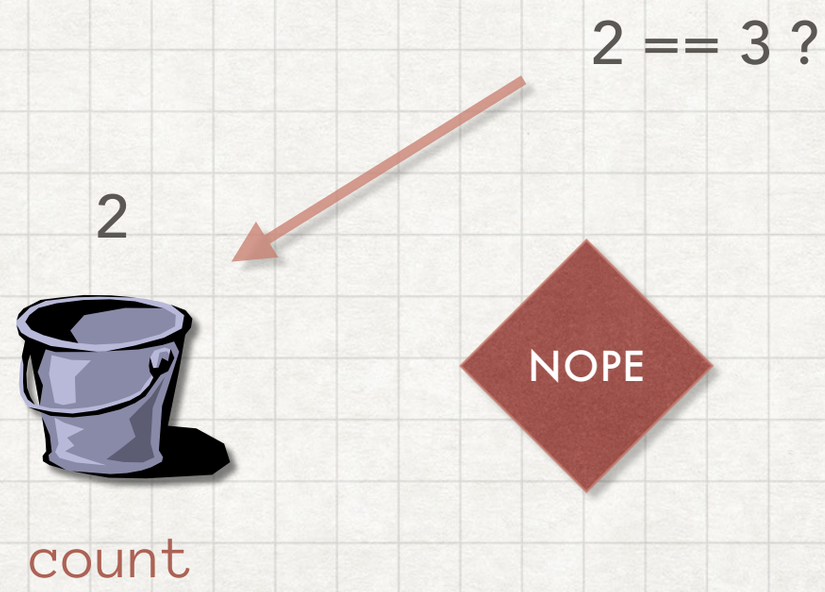
count

A WHILE LOOP IN JAVASCRIPT

Test the value in `count`. Is it three?

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. Say our count out loud.
   (In Javascript, we'll probably document.write() it.)
4. **Are we at 3 yet?**
   - **a. If not, go to Step 2.**
   - b. If yes, stop.

`count == 3 ??`

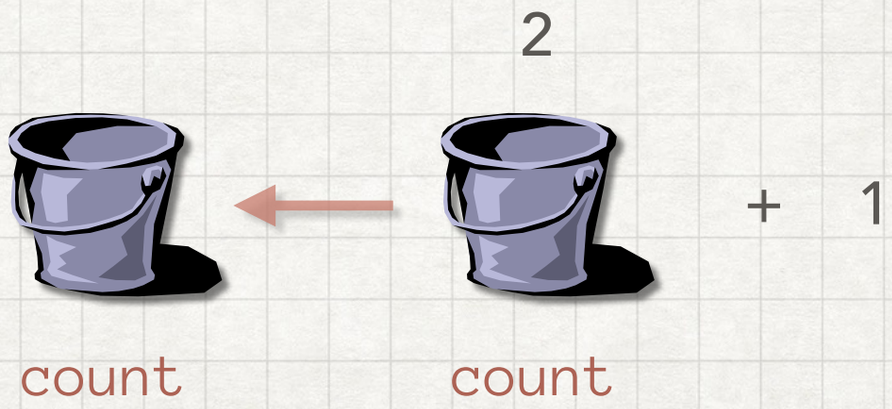


A WHILE LOOP IN JAVASCRIPT

We add one to our count. More precisely, we *increment the value in* `count`.

1. To begin, we haven't counted anything. We're at zero.
2. **Add 1 to our count.**
3. Say our count out loud.
   (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
   a. If not, go to Step 2.
   b. If yes, stop.

```
count = count + 1;
```



A WHILE LOOP IN JAVASCRIPT

We output the value in count somehow. Probably by printing it.

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. **Say our count out loud.**
   (In Javascript, we'll probably document.write() it.)
4. Are we at 3 yet?
   a. If not, go to Step 2.
   b. If yes, stop.

```
document.write(count);
```

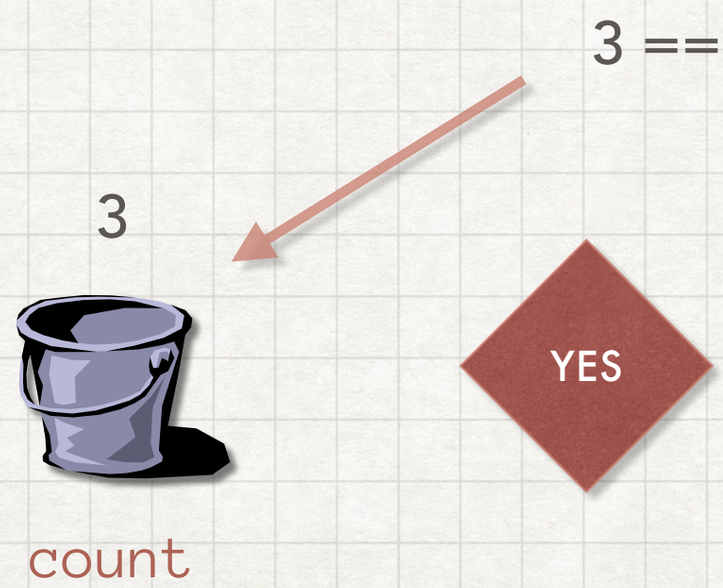
count

A WHILE LOOP IN JAVASCRIPT

Test the value in count. Is it three?

1. To begin, we haven't counted anything. We're at zero.
2. Add 1 to our count.
3. Say our count out loud.
   (In Javascript, we'll probably document.write() it.)
4. **Are we at 3 yet?**
   a. If not, go to Step 2.
   b. If yes, stop.

```
count == 3 ??
```



A WHILE LOOP IN JAVASCRIPT

In Javascript:

INITIAL STATUS

BOOLEAN TEST

DEGREE OF CHANGE

```
count = 0;
while (count < 3) {
    count = count + 1;
    document.write(count);
}
```

A WHILE LOOP IN JAVASCRIPT

In Javascript:

```
count = 0;
while (count < 3) {
    count = count + 1;
    document.write(count);
}
```

Notice that the keyword for the loop is *while*. That's important, because it means that the loop will continue to execute for as long as the boolean test continues to evaluate to true.

While the condition is true, the loop executes. When the test goes false, the loop will stop.

However, the test is only performed when Javascript executes the `while` test. It doesn't check the boolean on every line, just at the top of the loop.

ITERATING A LIST WITH WHILE

We can use a while loop to *iterate* (or visit every element) of a list:

```
letters = 'ABCDEFGH'.split("");

index = 0;
while (index < letters.length) {
    document.write(letters[index]);
    index = index + 1;
}
```

INITIAL STATUS

BOOLEAN TEST

DEGREE OF CHANGE

This uses the three components of a list we've seen. Remember that the index numbers start at zero, so the loop will fail when `index == letters.length`. In this list, after all, there is no element at `letters[8]`.

But Javascript has a special for loop that is built specifically to iterate lists and sequences.

Intro to Javascript

Loops

for

ITERATING A LIST WITH FOR

We can use a while loop to *iterate* (or visit every element) of a list:

```
letters = 'ABCDEFGH'.split("");

      INITIAL STATUS          BOOLEAN TEST          DEGREE OF CHANGE
for (index = 0; index < letters.length; index++) {
      document.write(letters[index]);
}
```

The for loop contains all three parts of a loop, but it compresses into a small, compact package.

IN

Javascript has a Boolean operator called in that checks for the presence or absence of a certain key in a JSON object. Notice that in checks *keys*, not values!

```
person = {
    'first': 'William',
    'last': 'Shakespeare',
    'birth': 1564,
    'death': 1616,
    'citizenship': 'English'
}
```

`'first' in person`	`True`
`'birth' in person`	`True`
`'Shakespeare' in person`	`False`
`1616 in person`	`False`
`'citizenship' in person`	`True`

FOR / IN

We can use this structure in a for loop to iterate an object's keys. Notice that we must use the **`object[key]`** syntax! The dot accessor syntax won't work here.

```
for (key in person) {
    document.write(person[key]);
}
```

Notice that we can do this with traditional arrays as well:

```
for (index in letters) {
    document.write(index);
}
```

SNEAK PREVIEW

Javascript also has another kind of loop called **`forEach()`**, but it behaves very differently. We need to learn about functions first before we can make it work.

```
letters = 'ABCDEFGH'.split("");

letters.forEach( stuff here );
```

Intro to Javascript

Conditionals and Loops