Booleans

Comparisons and Conditionals

Duration: 30 minutes
Q&A: 5 minutes by the end of the lecture

```
function abs (x) {
  // todo: write some code...



}
```

Let’s imagine we want to write a function that will calculate the **absolute value** of a number.

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
```

When we call our function, it will give us the non-negative value of whatever number was passed into our function.

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
```

When we call our function, it will give us the non-negative value of whatever number was passed into our function.

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```

When we call our function, it will give us the non-negative value of whatever number was passed into our function.

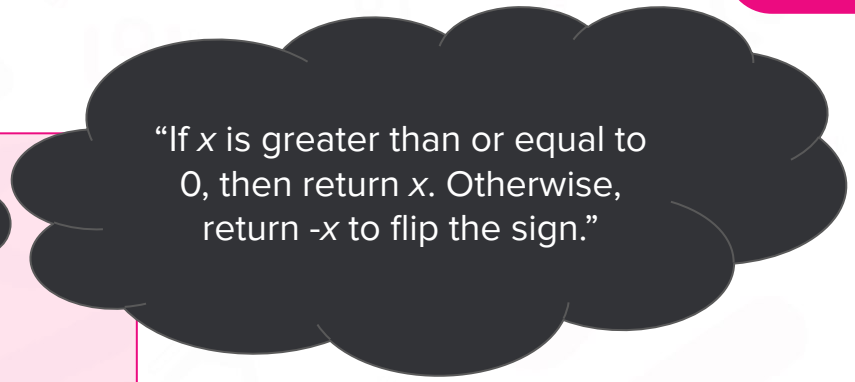
```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```

Q: How would we go about writing this function, using the tools that we've covered in this class so far?

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```

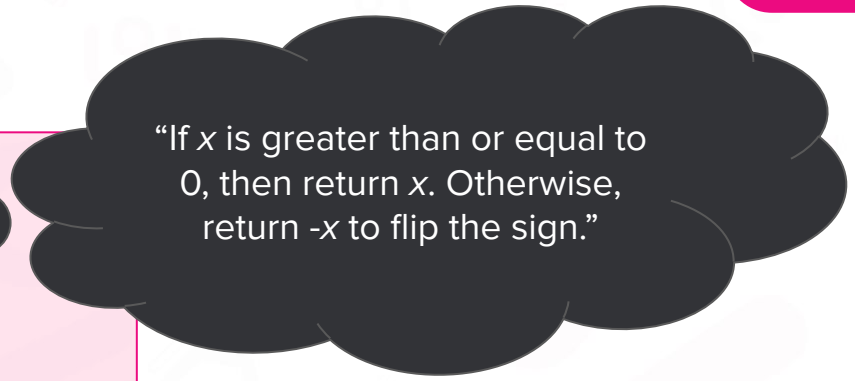
Hint: When writing a function that solves a problem, it's helpful to start by discussing a possible solution to the problem without writing code. How would you describe the process for finding the absolute value of a number?

```
function abs (x) {
  // todo: write some code..



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```



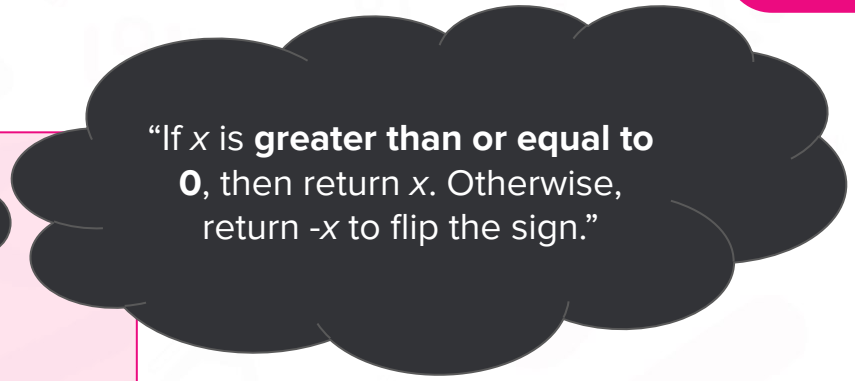
This explanation will do nicely. Now let’s reframe our original question.
Q: How can we translate this explanation into code, using the tools that we’ve covered in this class so far?

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```



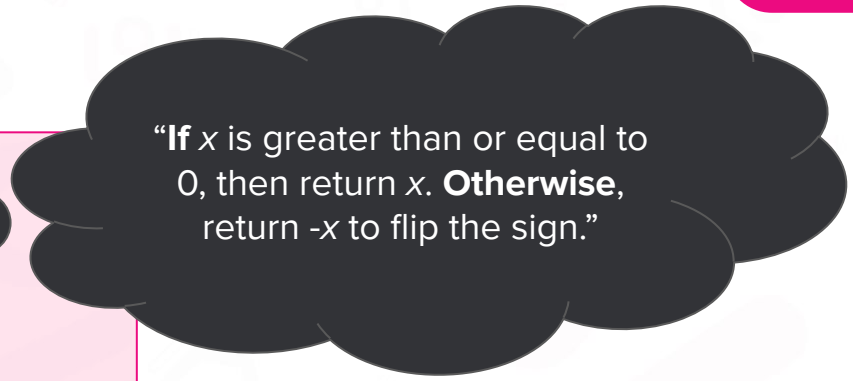
A: We can’t! Not yet, anyway.

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```



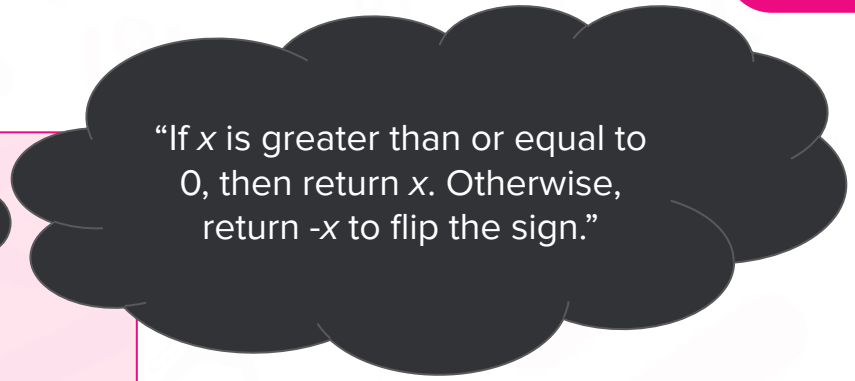
First, we haven’t yet discovered a way to determine if a number is greater than or equal to zero.

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```



Second, we haven’t learned how to do things **conditionally**.

```
function abs (x) {
  // todo: write some code...



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```



We’ll need to learn some new concepts before we can solve this problem. Let’s put it aside for now.

Booleans

Booleans are a data type that is used to represent *true* and *false* values.
Whereas strings and numbers have infinite possible values, Booleans can only be *true* or *false.*
Try typing the two boolean values into your JavaScript console.

Comparison Operators

A **comparison operator** will compare two values and return a boolean value indicating whether that expression evaluates to true or false.

```
3 > 5 // => false

9 < 10 // => true

‘hello’ === ‘world’ // => false
```

Comparison Operators

Here are some common mathematical comparison operators you will use when writing JavaScript:

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
===	equal to
!==	not equal to

Comparison Operators

These are comparison operators you should **never** use:

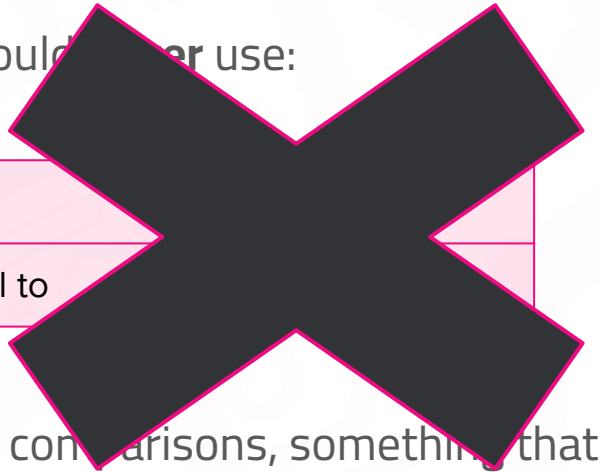
==	equal to
!=	not equal to

These operators perform loose-equality comparisons, something that is undesirable in almost all circumstances.

Comparison Operators

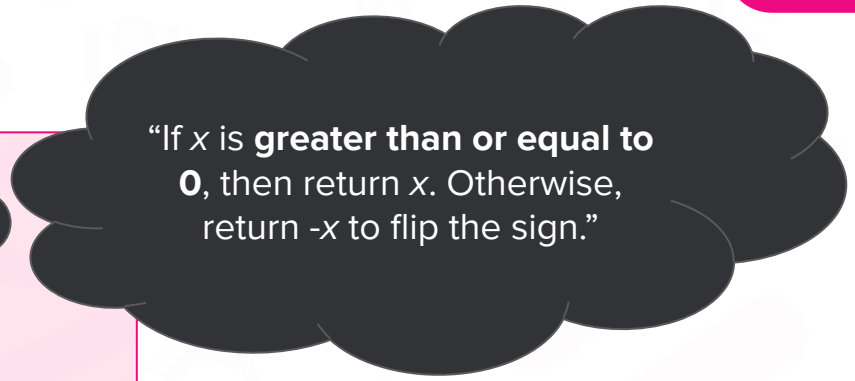
These are comparison operators you should never use:

==	equal to
!=	not equal to



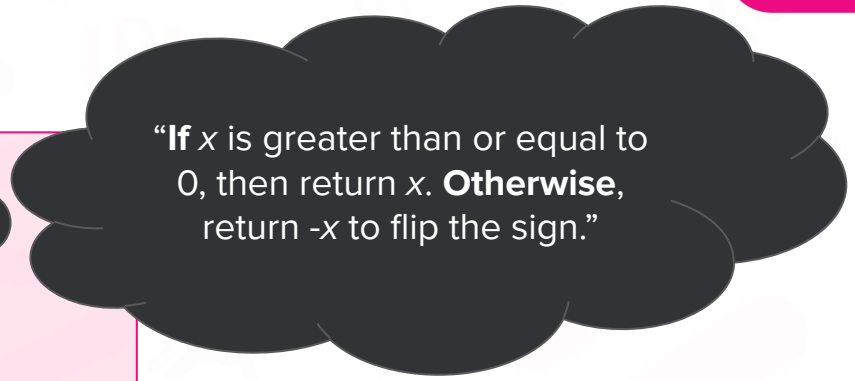
These operators perform loose-equality comparisons, something that is undesirable in almost all circumstances.

```
function abs (x) {
  // todo: write some code..



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```



Let’s briefly revisit our absolute value function. We now have the knowledge to determine whether or not *x* is greater than or equal to zero.

```
function abs (x) {
  // todo: write some code..



}

abs(1); // => 1
abs(-2); // => 2
abs(0); // => 0
```



Our question is posed in the form of "if some condition is met, then do this thing. Otherwise, do a different thing." We can represent this kind of question in code using **conditional statements**.

Conditional Statements

Conditional statements use the following syntax:

Conditional Statements

Conditional statements use the following syntax:

```
if ( <condition> ) { // 1. check the condition

}
```

Conditional Statements

Conditional statements use the following syntax:

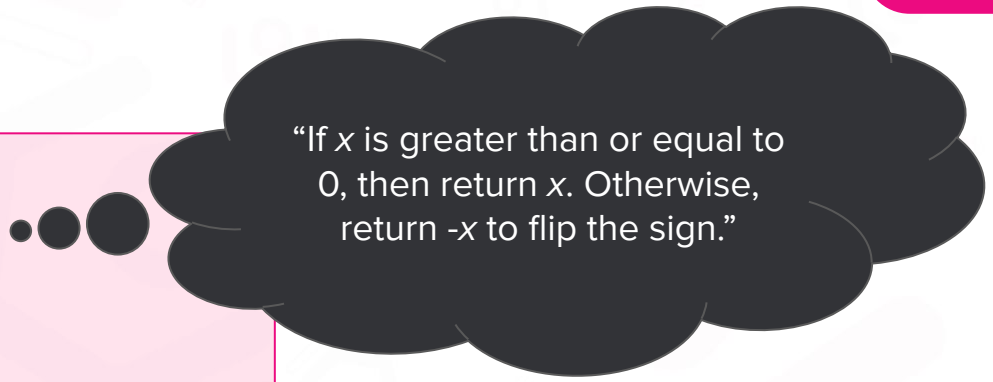
```
if ( <condition> ) { // 1. check the condition
  // 2. code that will only execute if <condition> is met
}
```

Conditional Statements

Conditional statements use the following syntax:

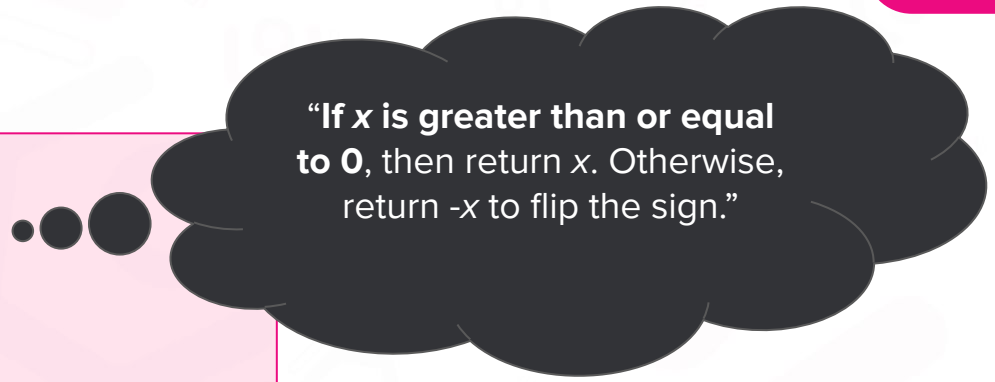
```
if ( <condition> ) { // 1. check the condition
  // 2. code that will only execute if <condition> is met
}

// 3. program execution continues
```

```
function abs (x) {



}
```



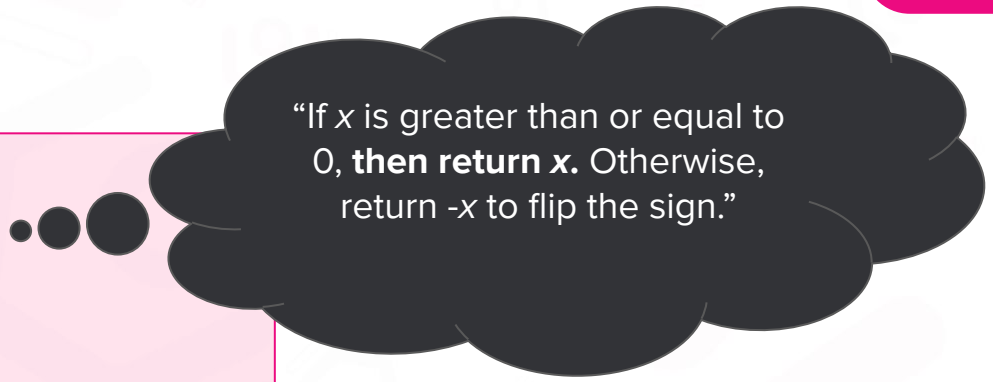
Now that we know how to compare values and write conditionals, let’s finish writing our absolute value function.

```
function abs (x) {
  if (x >= 0) {

  }

}
```



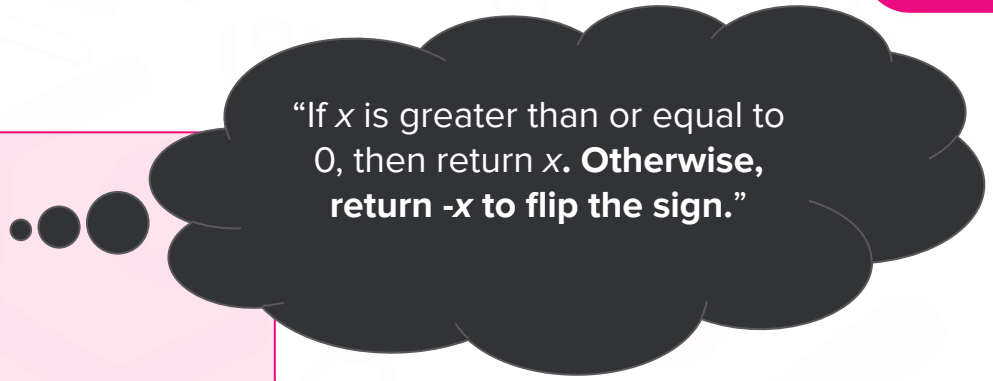
First we check our condition.

```
function abs (x) {
  if (x >= 0) {
    return x;
  }

}
```



Now we’ll add the code that should only be run when our condition is met.

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
    return -x;
}
```



Finally, we'll add some code outside of our conditional.

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}
```

We've completed our absolute value function, but there's some interesting syntax to examine here. **Q:** Can you identify something interesting about this function's syntax?

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}
```

A: This function has two `return` statements! We know that functions will only provide one value after performing some work. **How can we know what value this function will actually return?**

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32);
```

Let's discover the answer to this question by invoking `abs`. We'll step through this function line-by-line, examining it as though we were the computer executing this code. The pink arrow will indicate which line we're evaluating.

-32

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32);
```

We've invoked `abs` with an argument of `-32`. Recall that this value will be substituted inside our function every time we see the label `x` throughout this particular invocation of `abs`.

-32

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32);
```

We hit our conditional statement and must evaluate the expression inside. Is `-32` greater than or equal to `0`?

-32

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32);
```

It is not, so we will skip over all of the code inside our `if` statement.

-32

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32);
```

We reach a `return` statement. When the JavaScript interpreter encounters a `return` statement, it will evaluate the expression to the right of `return`, provide that value as the result of the function's work, and exit the function.

-32

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32);
```

In this invocation, that expression is `-(-32)`, which will evaluate to the positive numerical value `32`.

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32);
```

The interpreter returns to this expression in which we invoked `abs`, now evaluated to the value `32`.

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32); //=> 32
abs(24);
```

Let's add another invocation of `abs`, this time with a number that is already positive. What will happen differently in this second invocation?

24

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32); //=> 32
abs(24);
```

We enter into our function, now substituting the value 24 every time we encounter the label x.

24

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32); //=> 32
abs(24);
```

Our conditional statement will evaluate this expression: is `24` greater than or equal to `0`?

24

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32); //=> 32
abs(24);
```

This time our expression evaluates to `true`, so we must execute the code inside. We encounter a return statement. **Q:** What do you believe will happen next?

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32); //=> 32
abs(24); //=> 24
```

Remember that the return statement indicates that a function has finished running and should provide a value as its final result. The interpreter passes over any remaining code in the function.

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32); //=> 32
abs(24); //=> 24
```

After this exploration, we should have enough information to confidently answer our original question: **How can we know what value this function will actually return?**

```
function abs (x) {
  if (x >= 0) {
    return x;
  }
  return -x;
}

abs(-32); //=> 32
abs(24); //=> 24
```

The answer depends upon **which `return` statement is evaluated first**. Once a `return` statement is evaluated, the function will cease execution and return control of the program to the line on which the function was invoked.

CodingLab
Powered by RBK

That’s it