Comparisons

The backbone of most code is comparing one variable to another. For example, if you have a timer and you want it to countdown from 10 and stop at 0. Well, you need to know if the timer is currently greater than, less than, or equal to 0. If it's equal to 0, then we know we need to stop it.

Generic Comparisons

Take a look at these comparisons, and guess to see if they would be true or false.

```
let three = 3;
let two = 2;

// Greater than
three > two // true
two > two // false

// Greater than OR equal to
three >= two // true
two >= two // true

// Less than OR equal to
three <= two // false
two <= two // true

// Equal to (double equals)
three == two // false
two == two // true
// What would happen if we did this?
three = two // the variable three now equals to

// NOT equal to
three != two // true
two != two // false

// Strict equals is ===, meaning that they need to be the same value AND the same
type
let threeString = "3";

three == threeString // true
three === threeString // false

// Multiple comparisons in one
// && AND both sides have to be true
// || OR one of the sides have to be true
3 > 2 && 2 == 2 // true
3 > 2 || 2 == 2 // true
2 > 3 || 2 == 2 // true
2 > 3 && 2 == 2 // false
```

```
// If/Else Statements

if (three === threeString) {
    console.log("Both are equal");
} else if (3 > 2) {
    console.log("3 > 2");
} else {
    console.log("This shouldn't happen.");
}

if (currentTime <= 0) {
    stopTheTimer;
} else {
    keepCountingDown;
}
```

Comparisons Table

&& (AND) = if both conditions are true, then it will be true

TRUE && TRUE = TRUE

TRUE && FALSE = FALSE

FALSE && TRUE = FALSE

FALSE && FALSE = FALSE

|| (OR) = if at least one of the conditions is true, then it will be true

TRUE || TRUE = TRUE

TRUE || FALSE = TRUE

FALSE || TRUE = TRUE

FALSE || FALSE = FALSE