Comparison and Booleans

- Comparison and logical operators are useful in JS because they help us compare different conditions to one another. Comparison operators compare two values against one another and return a boolean value — either true or false.
- Comparisons in JavaScript can be made using <, >, <=, and >=, and work for both strings and numbers.

Comparison Operators	
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

- **Equality Operators**
- Now let's take a look at equality operators.
- Equality operators check to see whether two values are the same as, or equal to, one another.
- **Equality** (===): This operator will accept any two types of data as inputs and (just like the comparison operators) evaluate to a Boolean value. It will only evaluate true if both sides are completely identical in data type and value.
- *For example*: 5 === 5 will evaluate to true, while 5 === '5' will evaluate to false since, while the values are the same, 5 is a number and '5' is a string.
- **Inequality** (!==): This operator will also accept any two types of data as inputs and evaluate to a Boolean value. It is essentially the reverse of the equality operator — it compares two values to check that either the data type or value are *not* the same.
- *For example*: 5 !== 5 will evaluate to false, while 5 !== '5' will evaluate to true.

Test Yourself

Type each command given in this JS Bin Console.

Before you press enter, take a moment to think about what value the console will return.

```
8 > 8
8 >= 8
8 < 8
8 < 13
8 <= 15
7 === 7
7 === "7"
7 !== 7
7 !== "7"
6 === 7
6 !== 7
```

- **Null and Undefined**

- At this point, we've covered most of what you need to know about basic expressions.
- However, there are a few quirks and exceptions that we've (until now) glossed over, especially related to Boolean logic. Let's take a closer look at a few.

Undefined

Say you've just opened a JavaScript console. You want to define a new variable but aren't sure what the value is *just yet*.

Example : var someData;

Without **assigning** a value to a variable, that variable becomes undefined. We can see this in real time:

```
var someData;
console.log(someData);
// => 'undefined'
```

Undefined

One way to check to see if a variable is undefined is to use typeof. This method is possible because undefined is a specific object and its own data type.

```
// we need a new variable...
var anotherData;

typeof anotherData;
// => "undefined"
```

Null

null values are values that you decide have **no value**. Why would you want to do this? Why not use undefined?

Convention is that undefined is reserved for variables whose values haven't been set yet. null is reserved for variables whose value is explicitly nothing — instead of just "not defined yet."

null gives us a way to "reset" the value of a variable to "nothing."

- **Null**
- Here's an example:
- Suppose you have an application for keeping track of your possessions. You might have a string called locationOfKeys indicating where you can find your keys.
- Then, one day, your keys get lost. What's the value of locationOfKeys now? Well, it's "nothing" — they are lost.
- If null didn't exist, we would have to invent a special string value (perhaps "lost") to signify that the keys are missing.
- null gives us a standard way of handling that kind of situation in which we can simply say locationOfKeys = null;

Null

That's the purpose of null. It is designed to represent the *lack of a value*.

Whenever variables are defined without any value, they are undefined. This can become tricky to troubleshoot over time, as it acts as a catchall for *everything without a value*.

Null

We can **specify** our variables as `null` to represent that there is no data.

```
// we will define a variable with no value, or null.
var playerScore = null;
```

We can then **evaluate** if our value is `null`:

```
playerScore === null // The player has not scored anything
```

Null

What if we are provided with an input representing a player's actions in a timed game?

In *Dance, Dance, Revolution*, a player must perform an action at a precise interval. It streams a set of commands a player must *dance* to. These are timed to music.

The actions stream across the screen, and, as they pass by the middle of the screen, the player *must* perform an action in order to score a point. The game can evaluate the player's score in real time!

```
var userInput = null;
userInput === null // No points... this time
```

Boolean Logic

Everything in JavaScript — from the strings we learned about in Unit 1 to the null and undefined values we just covered — has an inherent Boolean value that can be thought of as being either *truthy* or *falsey*.

But what does it mean to say that, for example, "apple" is *truthy*?

"apple" is not literally true, but the javascript language considers it to be *truthy*.

We can prove this by "double negating" a value in javascript, to force (or "coerce") it into its boolean value.

Boolean Logic

To do this, we will use a new operator - the NOT operator.

NOT(!): If the value is truthy, return false; if the value is falsey, return true.

A handy little trick is that we can put !! before any value to check to see if it is *truthy* or *falsey*.

Take a look at the code below:

```
true //=> true
!true //=> false
!!true //=> true
// Therefore true is truthy!

"apple" //=> "apple"
!"apple" //=> false
!!"apple" //=> true
// therefore "apple" is truthy!
```

Falsey

Something is *falsey* when it can be coerced into the Boolean value false. The *falsey*category of values includes:

- false
- 0 (zero)
- "" (empty string)
- null
- undefined
- NaN (a special Number value meaning "Not a Number"!)

Truthy

Everything else in JavaScript is *truthy*.

Something is *truthy* when it can be coerced into the Boolean value true.
In JavaScript, *truthy* values include:

- "abc" (any non-empty string)
- -1, 1, 2.5 (any non-zero number)
- true

Summary

Below are the exact rules Boolean operators follow when dealing with non-Boolean input values.

falsey	*truthy*
`false`	`true`
`0`	All numbers except `0`
Empty Strings (`""`)	All non-empty strings
`undefined`, `null`, and `NaN` (Not a number", as special type of numeric value)	Pretty much everything else

Test Yourself

Type each command given in this JS Bin Console.

Before you press enter, take a moment to think about what value the console will return.

1. "orange"
2. !!"orange"
3. 7
4. !!7
5. false
6. !!false
7. true

- Success is prepaid.
  - -Unknown