# Comparison and Ternary Operators

You'll look at how comparison and ternary operators are used to evaluate conditions and boolean values.

Let's see some booleans values. Booleans are either true or false.

## Comparison Operators

We can compare two numbers with `>`, `>=`, `==`, `===`, `<=`, `<`. We will discuss the difference between `==` and `===` soon. For the rest of the operators, the result of the comparison is a boolean.

### The ! operator

The `!` operator negates its operand. True becomes false, and false becomes true.

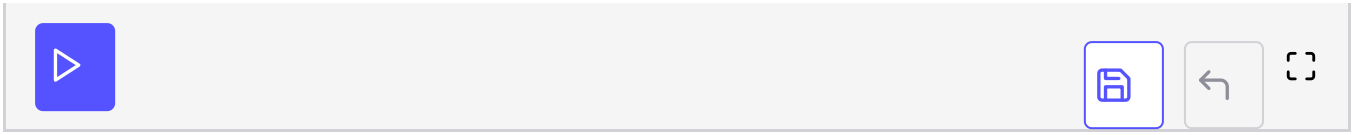
> *A **truthy** value is a value `v` for which `!!v` is `true`. Examples of truthy values are: nonzero integers, strings containing at least one character.*
> *A **falsy** value is a value `w` for which `!!w` is `false`. Examples of falsy values are: empty string, `0`, `null`, `undefined`.*

We can convert a value to a boolean by negating it twice: `!!`:

- assuming `v` is truthy, `!v` becomes `false`. `!!v` becomes `!false`, which becomes `true`.
- assuming `w` is falsy, `!w` becomes `true`, and `!!w` becomes `false`.

```
console.log("5 <= 5", 5 <= 5);
console.log("5 < 5",5 < 5);
console.log("!(5 < 5)",!(5 < 5));   // ! stands for negation. !true = false, !false = true.
console.log("!!\"\"",!!"")
console.log("!!\"a\"",!!"a")
console.log("!!0",!!0)
console.log("!!1",!!1)
console.log("!!NaN",!!NaN)
console.log("!!Infinity",!!Infinity)
console.log("!!null",!!null)
```

```
console.log("!!undefined",!!undefined)
```



### Equal to

For values `a` and `b`, `a == b` is true if and only if both `a` and `b` can be converted to the same value via type casting rules. This includes:

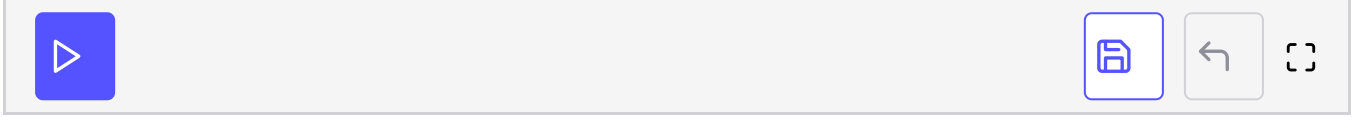
- `null == undefined` is true
- If an operand is a string and the other operand is a number, the string is converted to a number
- If an operand is a number and the other operand is a boolean, the boolean is converted to a number as follows: `true` becomes `1`, and `false` becomes `0`.

Don't worry about the exact definition, you will get used to it.

### Equal Value and Equal Type

For values `a` and `b`, `a === b` is `true` if and only if `a == b` and both `a` and `b` have the same types.

```
console.log("5 == \'5\'",5 == '5');   // '5' is converted to 5
console.log("5 === \'5\'",5 === '5');  // types have to be the same
console.log("0 == \'\'",0 == '');    // '' is converted to 0
console.log("0 === \'\'",0 === '');   // types have to be the same
console.log("NaN == NaN",NaN == NaN); // I know... just accept this as something odd and funr
```

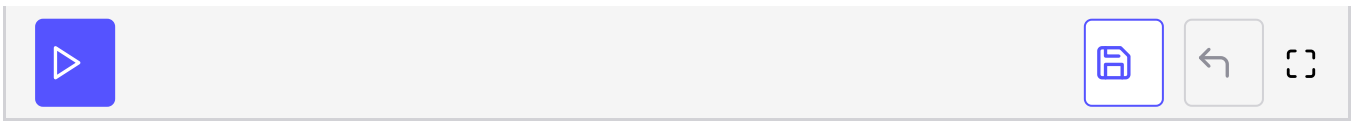


### Not equal to

The negation of `==` is `!=`. Read it as *is not equal to*.

The negation of `===` is `!==`.

```
console.log("5 != \'5\'",5 != '5')
console.log("5 !== \'5\'",5 !== '5')
```



## Ternary Operator

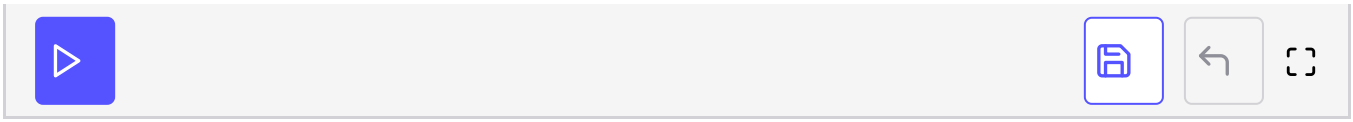
Let me introduce the ternary operator to drive home a strong point about truthiness.

The value of `a ? b : c` is:

- `b` if `a` is truthy
- `c` if `a` is falsy

It is important to note the difference between `2 == true` and `!!2`.

```
console.log("2 == true",2 == true)    // true is converted to 1
console.log("!!2",!!2)          // 2 is a truthy value
console.log("2 == true ? \'the condition is true\' : \'the condition is false\'")
console.log(2 == true ? 'the condition is true' : 'the condition is false')
console.log("!!2 ? \'the condition is true\' : \'the condition is false\'")
console.log(!!2 ? 'the condition is true' : 'the condition is false')
```



I have seen the nastiest bug in my life in a code, where a condition was in a format `num == true`. As I never felt like learning boring definitions, my lack of knowledge shot me in the foot, because I assumed the opposite conversion in `2 == true`. I can save you some headache by highlighting this common misconception. In `2 == true`, `true` is converted to `1`, and not the other way around.