noble desktop

UNIT 02

LESSON 02.01

JavaScript BOOTCAMP

logical operators

conditional logic: if-else

if - else if - else

variable scope: let vs var

A comparison operator *compares* two values and resolves to a boolean (**true** or **false**). The various operators are:

- **==** equal to in value
- **===** equal to in both value and data type
- **!=** not equal to in value
- **!==** not equal to in value or data type or both
- **>** greater than in value
- **<** less than in value
- **>=** greater than or equal to in value
- **<=** less than or equal to in value
- **!** not prefix (opposite)

single vs. double vs. triple equal signs

- **=** assigns a value:
  - **x = 5**. Whatever **x** used to be, now it's 5.
- **==** *compares* two values; it does *not* assign a value.
  - returns **true** if the two values are equal
  - returns **false** if the two values are not equal.
  - does not consider datatype. **5 == "5"** is **true**.
- **===** compares two values; it does *not* assign a value.
  - returns **true** if they are equal in both value *and* datatype.
  - returns **false** if they are not equal in either value *or* datatype.
  - considers datatype: **5 === "5"** is **false**.
- **!=** checks for inequality of value, but not of datatype.
- **!==** checks for inequality of both value and data type.

1. Open the console and type these comparisons, hitting enter each time:

```
8 == 8; // true
8 == "8"; // true
8 === "8"; // false
8 >= "8"; // true
```

2. Make some comparisons using the inequality operators:

```
7 != 8; // true
7 != 7; // false
7 != '7'; // false
7 !== '7'; // true
```

3. Make some comparisons using the greater than less than equal to operators:

```
7 >= 8; // false
7 >= '7'; // true
7 <= 8; // true
7 <= '7'; // true
```

conditional logic with if else

An **if-statement** makes a comparison between two values, which resolves to a **boolen**. If the condition is **true**, the code inside the curly braces of the if statement will run. If the condition is **false**, the code will not run. There may be an "else part" that runs if the condition is false.

How to write an **if-statement()**:

- write *if* followed by a pair of parentheses: **if()**
- inside that put the condition to evaluate: **if(5 > 3)**
- after that, put a pair of curly braces:
  - **if(5 > 3) { }**
- inside the curly braces, write the code that you want to run if the condition is true:
  - **if(5 > 3) { console.log("It's true!"); }**

4. Do some comparisons inside if-statements:

```
if(7 == "7") {
    console.log('close enough');
}

if(7 === "7") {
    console.log('close but no cigar');
}
```

A boolean is already true or false, so we don't need to explicitly compare it true or false.

5. Do two versions of an if-statement with a boolean, with and without parentheses:

```
let raining = true;

if(raining == true) {
    console.log("It's raining, so bring an umbrella!"); // runs
}

if(raining) { // also works
    console.log("It's raining, so bring an umbrella!"); // runs
}
```

if else

What if we want go to the beach if it *not* raining. For that, we need an **else** part":

6. Make raining *false* and add an *else*:

```
raining = false;

if(raining) {
    console.log("It's raining, so bring an umbrella!");
} else {
    console.log("It's not raining! Let's go to the beach!"); // runs
}
```

Strings can also be evaluated with if-else logic.

7. Check if the weather is "sunny":

```
let weather = "sunny";

if(weather == "sunny") { // runs
    console.log("It's sunny! Don't forget your sunglasses!");
} else {
    console.log("It isn't sunny! Leave the sunglasses at home!");
}
```

With "number-like strings", the digits are evaluated like individual letters. "50" is greater than "100", because 5 is greater than 1.

8. Compare two "number-like strings":

```
let full = "100";
let half = "50";
```

```
    if(full > half) {
        console.log('"100" > "50"');
    } else {
        console.log('because 5 > 1'); // runs
    }
```

9. Convert the "number-like strings" to actual numbers and try again:

```
    full = Number("100");
    half = Number("50");

    if(full > half) {
        console.log('100 > 50'); // runs
    } else {
        console.log('50 > 100');
    }
```

if-else if-else logic

else if adds another condition to evaluate if **if** returns false:

10. Try this **else if**:

```
    let highScore = 15000;
    let myScore = 10000;

    if (highScore > myScore) {
        console.log('You did not beat the high score!');
    } else if (highScore < myScore) {
        console.log('You beat the high score!');
    } else {
        console.log('You tied the high score!');
    }
```

11. Try different values of **myScore** to get all three outcomes.

12. Check the weather with **else if** logic:

```
    weather = "cloudy";

    if (weather == "rainy") {
        console.log('Go to the museum!');
    } else if (weather == "sunny") {
        console.log('Go to the beach!');
    } else {
        console.log('Go to the park!');
    }
```

13. Try different values of **weather** to get all three outcomes.

There can be any number of **else if** blocks between the opening **if** and the closing **else**.

14. Run this code with its three **else if** clauses. The best approach is to go higher to lower by ranges (90+, 80-89, 70-79, 65-69, 64 and below):

```
let score = 77;
let grade = "";

if(score >= 90) {
    grade = "A";
} else if(score >= 80) {
    grade = "B";
} else if(score >= 70) {
    grade = "C";
} else if(score >= 65) {
    grade = "D";
} else {
    grade = "F";
}
let reportCard = **Your score: ${score}.
Your grade: ${grade}.**
console.log(reportCard);
```

block scope

As you recall from a previous lesson:

let is **block-scoped**, meaning it is available only inside the the curly braces in which it was declared. Block-scoping prevents variables from "leaking out" into parts of the application where they don't belong.

The curly braces of if-statements enclose code blocks, so **let** variables declared therein are confined to that code block. This is not the case with **var**, which is global in scope even when it is declared within curly braces.

15. Write an "if-statement" with a **let** variable declared inside its curly braces. Declare the variable inside and outside the code block:

```
let meal = "lunch";

if (meal == "lunch") {
    let special = "Burritos";
}

console.log(meal); // lunch
console.log(special); // ERROR: special is not defined
```

We get **not defined**, because **special** is block-scoped to its if-statement; **special** does not exist in the global scope where we are attempting to access it.

16. Run the same test, but with **var**:

```
    if (meal == "lunch") {
        var special = "Burritos";
    }
    console.log(special); // Burritos
```

The "var variable" still exists after the if-statement, because, **var** is in the **global scope**, even though it was declared inside an "if-statement".

let instead of var for better scope control

This is why **let** is preferred over **var**. With **let**, you have better control over variable scope. With **var**, variables "leak out" from their blocks into the rest of the application.

global variables with let

With **let**, you can still have **global variables**. Just declare them outside of any curly braces, and they will be available throughout the script.

- **END Lesson 02.01**
- **NEXT Lab 02.02**
- **Lesson 02.02**