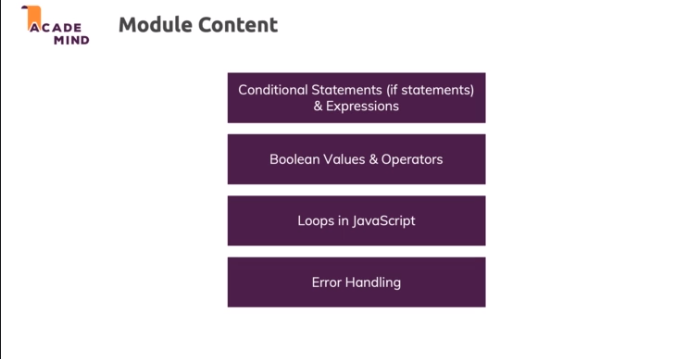
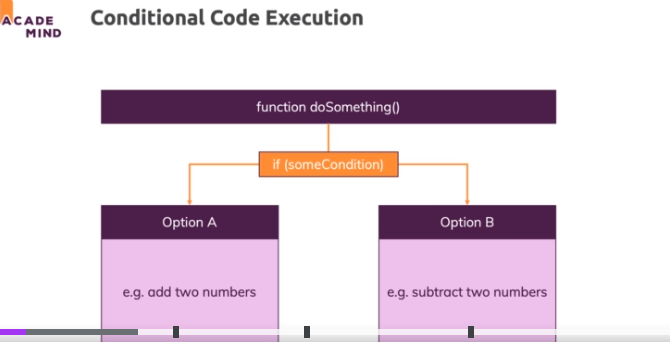
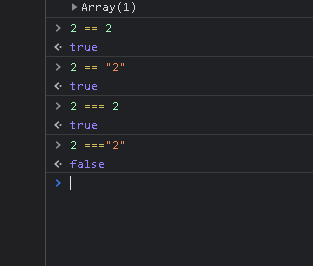
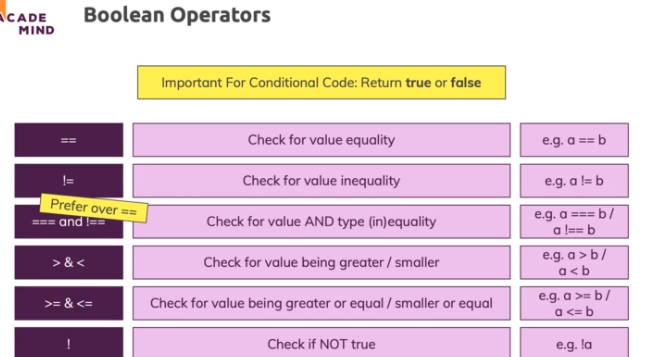
**Working with ControlStructures (if Statements, Loops, …Error handling)**



**Introducing "if" Statements & Boolean (Comparison) Operators**







**Using Booleans in Conditions & More on Text Comparisons**

    1. if (condition) { ... }

simply has to be a **boolean value**.

Often, you'll **generate** such a boolean value with the help of ===, >, < etc. **All these operators yield boolean values** (without changing the variables/ values you're using them on).

Since if only wants a boolean, you of course **don't have to use such an operator**. If you already got a variable that holds a boolean, you can use it without any extra operator.

Example:

    1. const isLoggedIn = true;

    2. if (isLoggedIn) {

    3.     // This code will execute because isLoggedIn is true => A valid condition

    4. }

You could write

    1. const isLoggedIn = true;

    2. if (isLoggedIn === true) {

    3.     ...

    4. }

but that would be **redundant**. You'd generate another new boolean where you already got one.

You can use the ! operator to negate ("invert") the value:

    1. const isLoggedIn = true;

    2. if (!isLoggedIn) {

    3.     // This code will NOT execute because isLoggedIn is true but ! inverts it (in this check)

    4. } else {

    5.     // This would execute because !isLoggedIn yields false => else block executes

    6. }

Again, that would be similar to:

    1. const isLoggedIn = true;

    2. if (isLoggedIn !== true) {

    3.     // This would NOT execute

    4. } else {

    5.     // This would execute because isLoggedIn is true and hence !== true yields false

    6. }

But again, that would be redundant.

More on Text (String) Comparisons

Strings can also be compared with greater than (>) or lower/ smaller than (<) operators.

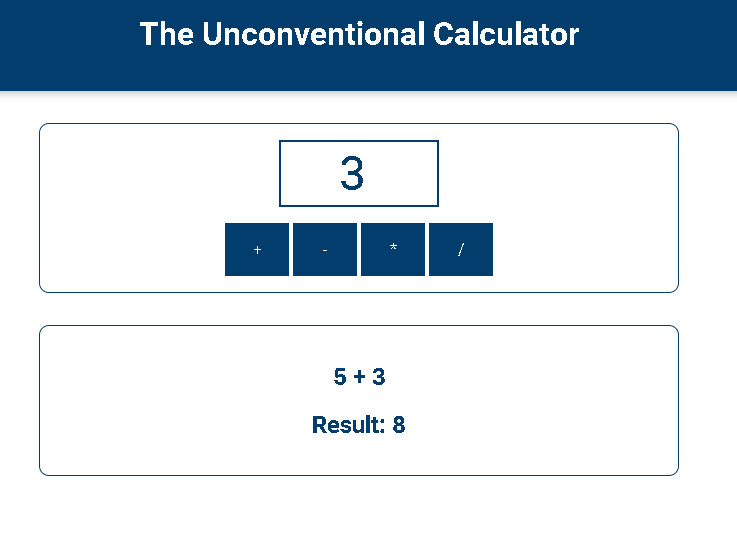
JavaScript compares strings based on standard lexicographical ordering, using Unicode values.

That means that b is greater than a for example.

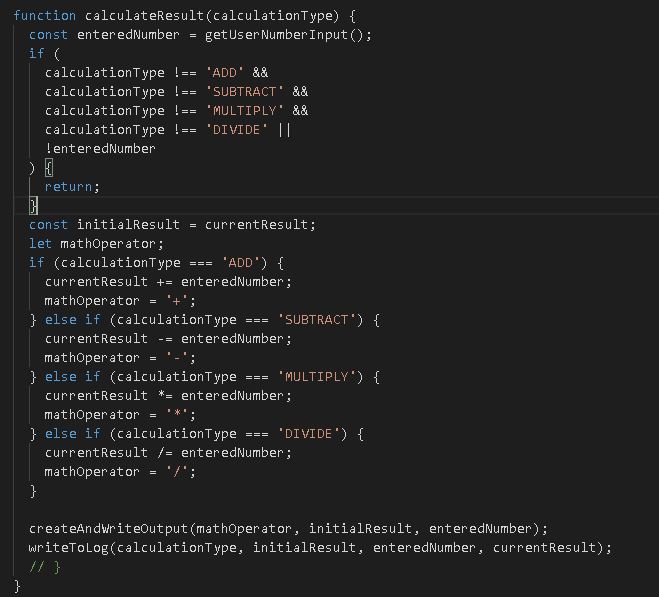
JavaScript always looks at the first character and only considers other characters if the first character is similar. In addition, capital characters are considered to be smaller than lowercase characters.

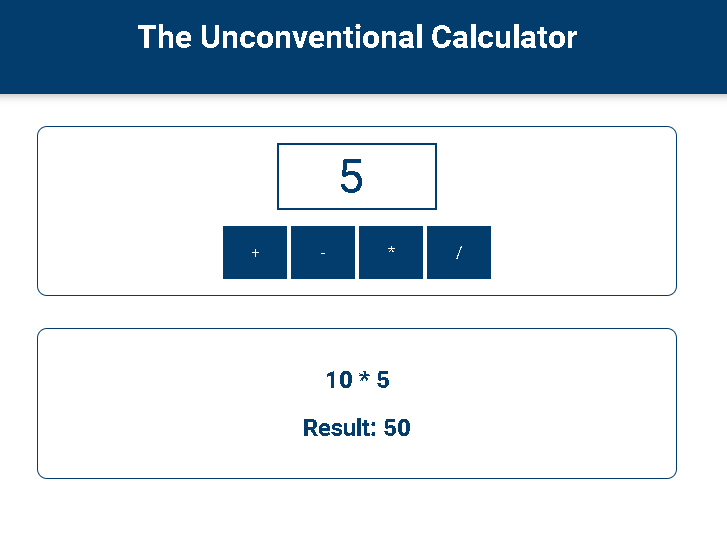
**Using "if" Statements**



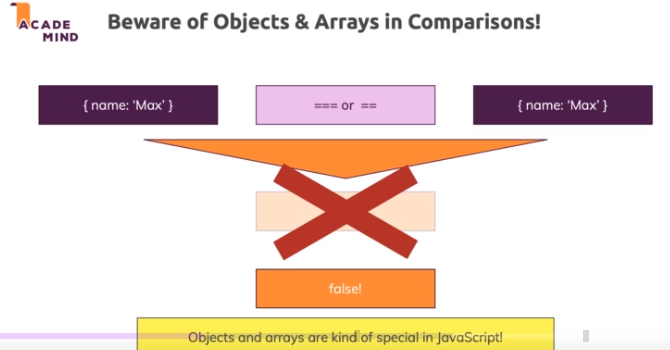


**Working with "if", "else" and "else-if"**

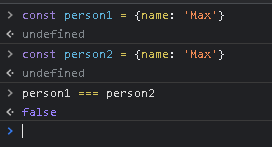


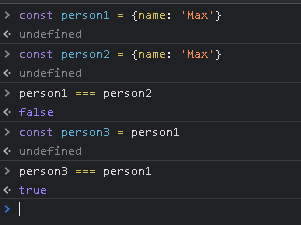


**Beware When Comparing Objects & Arrays for Equality!**

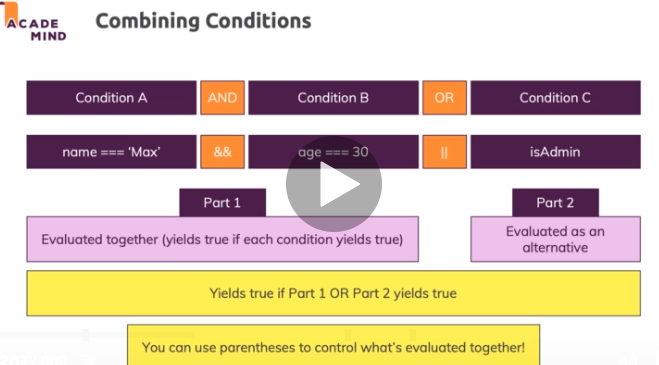


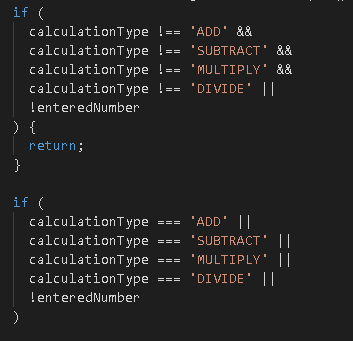
Two objects stored in different variables will never be the same



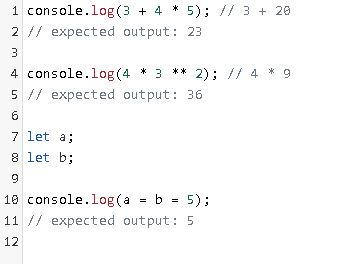


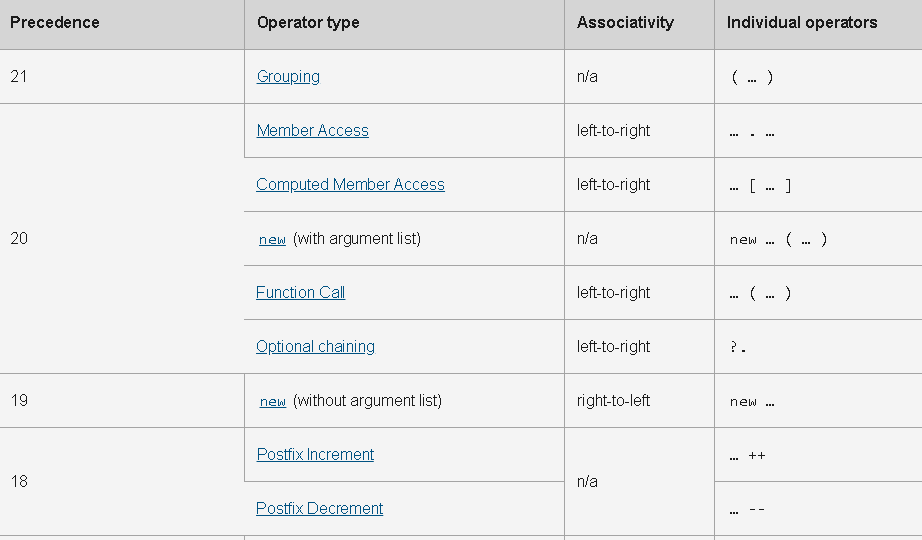
**The Logical AND and OR Operators**



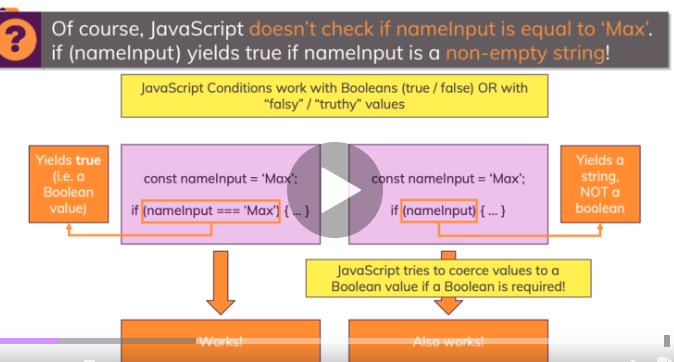


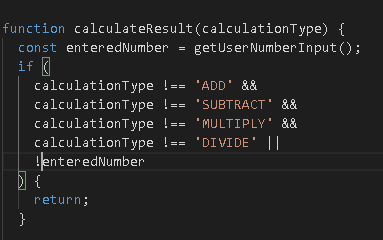
**Understanding Operator Precedence**

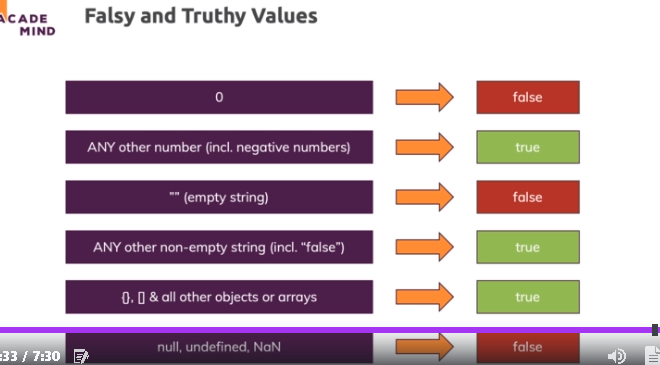




**Beyond true/ false: "Truthy" and "Falsy" Values**







**Coercion vs Conversion**

It's important to understand that JavaScript is able to use variables in conditions - even without comparison operators.

This is kind of obvious, if we consider a boolean variable, for example:

    1. let isLoggedIn = true;

    2. if (isLoggedIn) {

    3.     ...

    4. }

Since if just wants a condition that returns true or false, it makes sense that you can just provide a boolean variable or value and it works - without the extra comparison (if (isLoggedIn === true) - that would also work but is redundant).

Whilst the above example makes sense, it can be confusing when you encounter code like this for the first time:

    1. let userInput = 'Max';

    2. if (userInput) {

    3.     ... // this code here will execute because 'Max' is "truthy" (all strings but empty strings are)

    4. }

JavaScript tries to coerce ("convert without really converting") the values you pass to if (or other places where conditions are required) to boolean values. That means that it tries to interpret 'Max' as a boolean - and there it follows the rules outlined in the previous lecture (e.g. 0 is treated as false, all other numbers are treated as true etc.)

It's important to understand that JavaScript doesn't really convert the value though.

userInput still holds 'Max' after being used in a condition like shown above - it's not converted to a boolean. That would be horrible because you'd invisibly lose the values stored in your variables.

Instead,

    1. if (userInput) { ... }

is basically transformed (behind the scenes) to

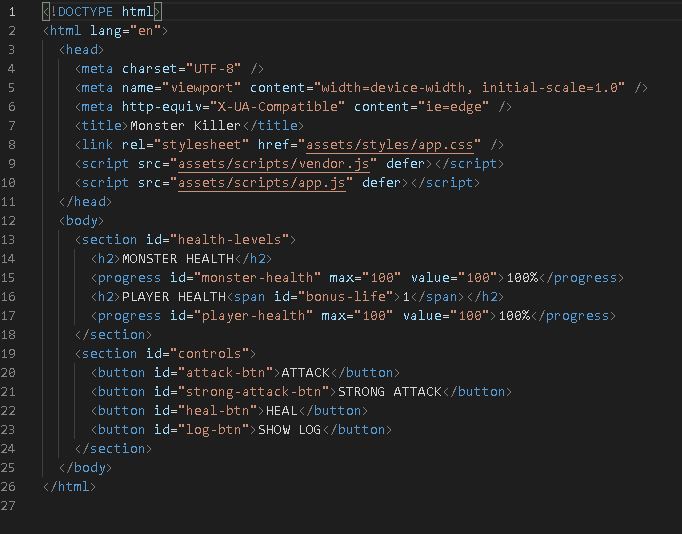
    1. if (userInput === true) {

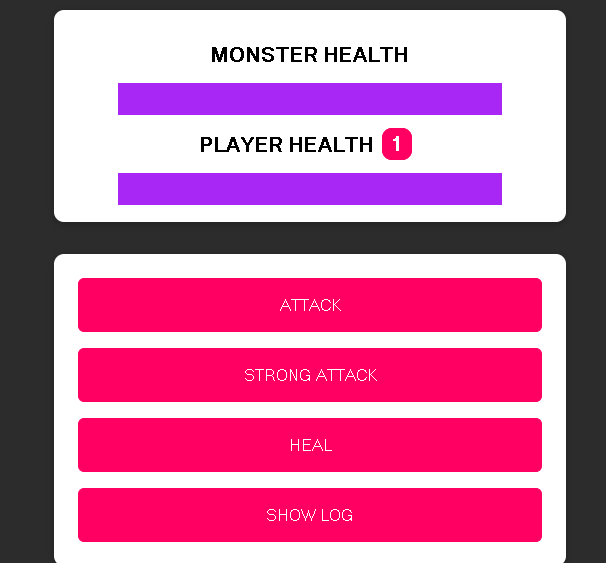
And here, the === operator generates and returns a boolean. It also doesn't touch the variable you're comparing - userInput stays a string. But it generates a new boolean which is temporarily used in the comparison.

And that's exactly what JavaScript automatically does when it finds something like this:

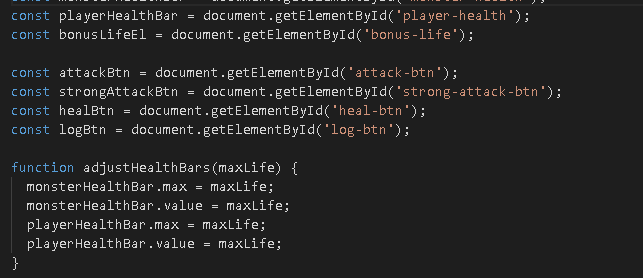
    1. if (userInput) { ... }

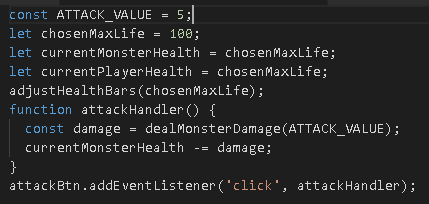
**Setting Up a Bigger Example Project (The "Monster Killer")**

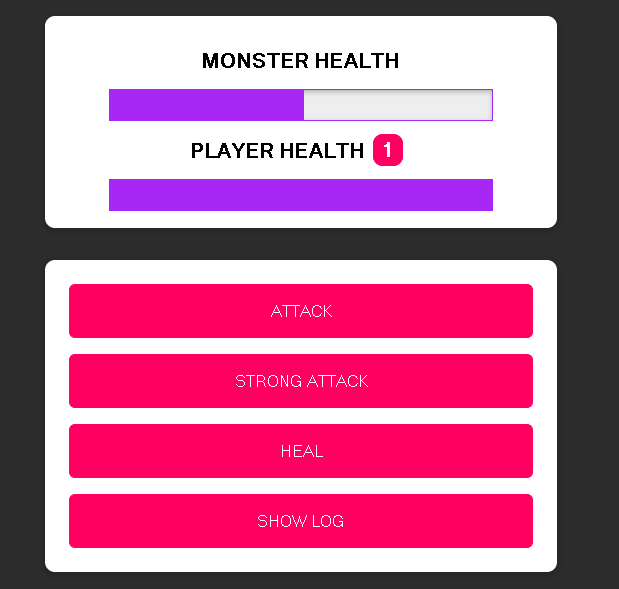




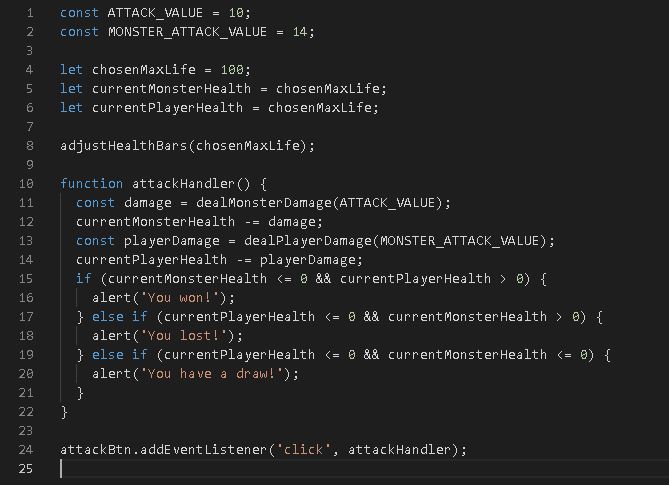
**Adding an "Attack" Function**

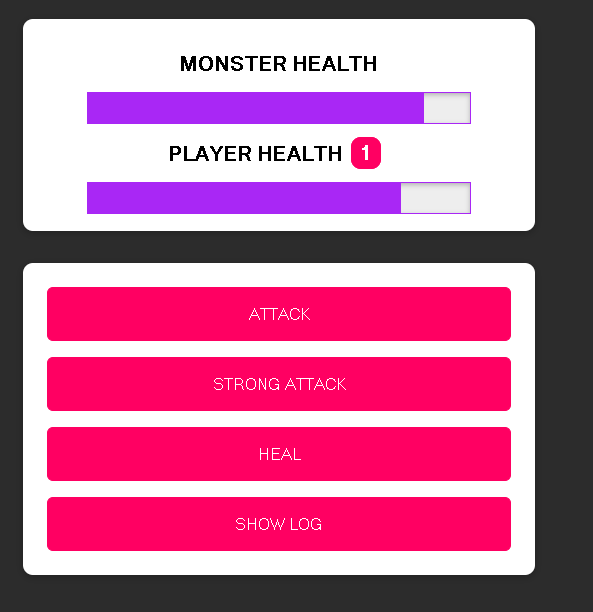




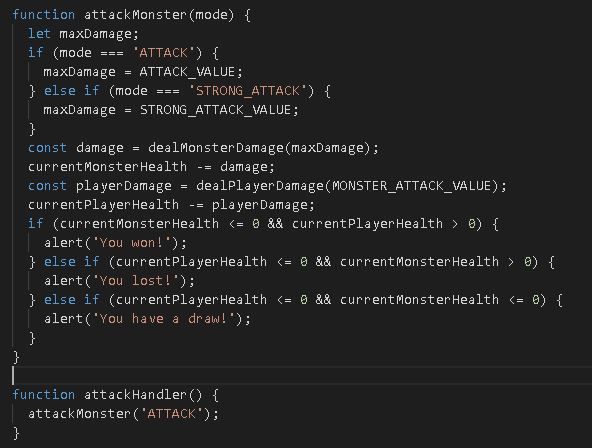


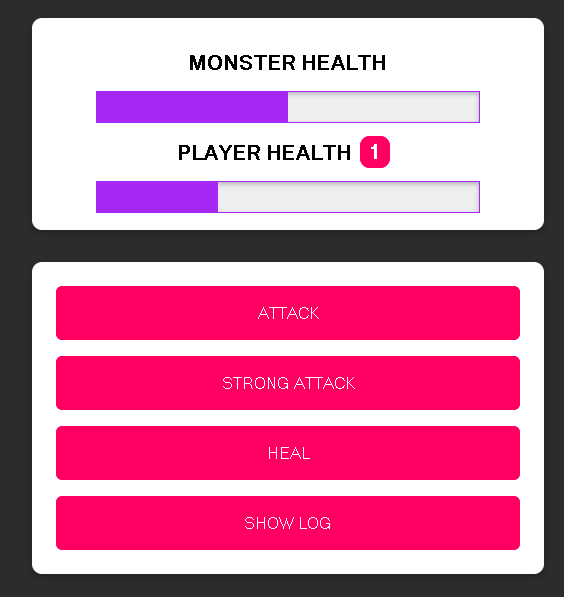
**Using "if" Statements for Checking the Win-Condition**





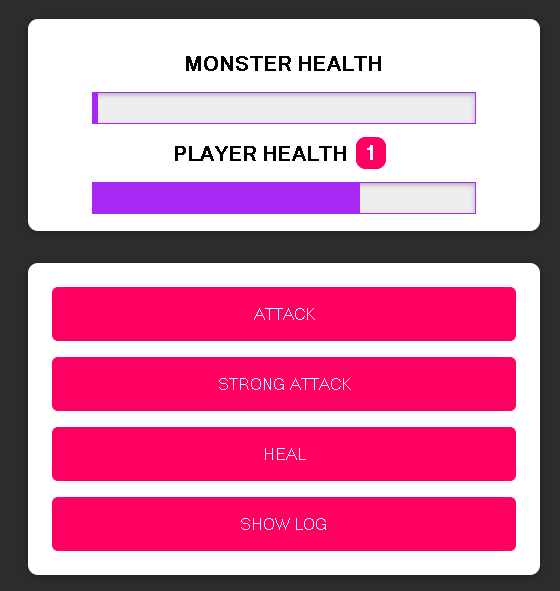
**Adding More "if" Statements & A "Strong Attack" Functionality**



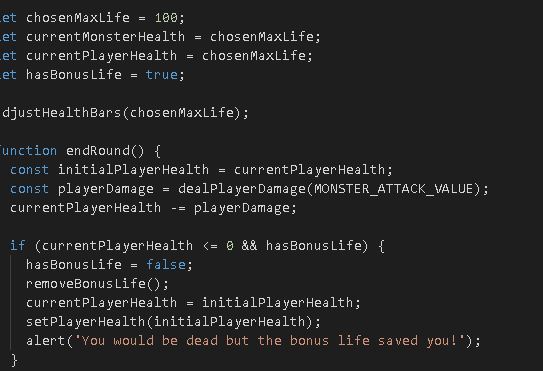


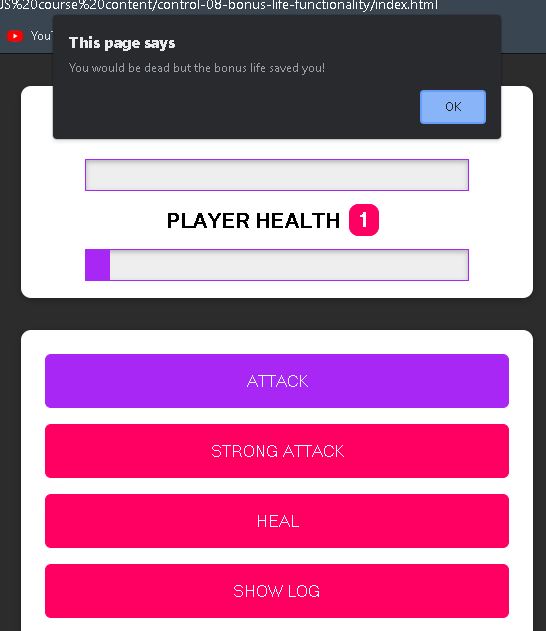
**Time for a "Heal Player" Functionality!**



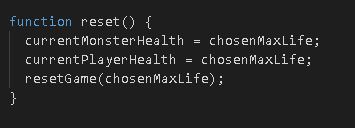


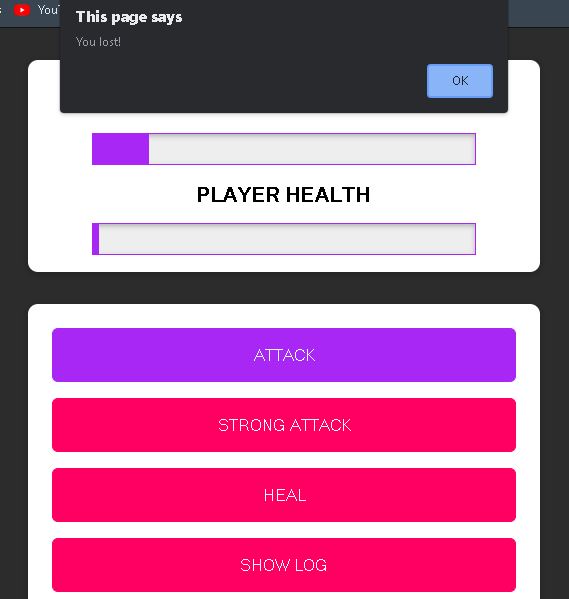
**Controlling the Conditional Bonus Life (Without Boolean Operators!)**



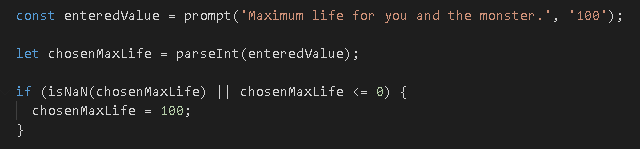


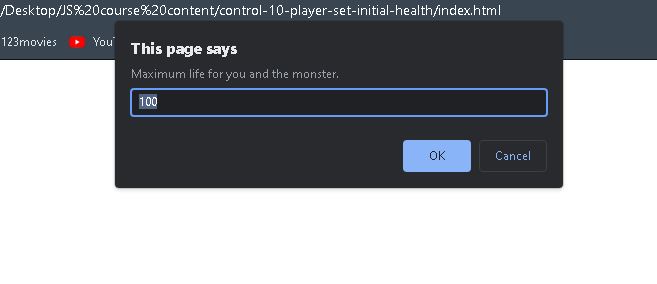
**Adding a "Reset Game" Functionality**



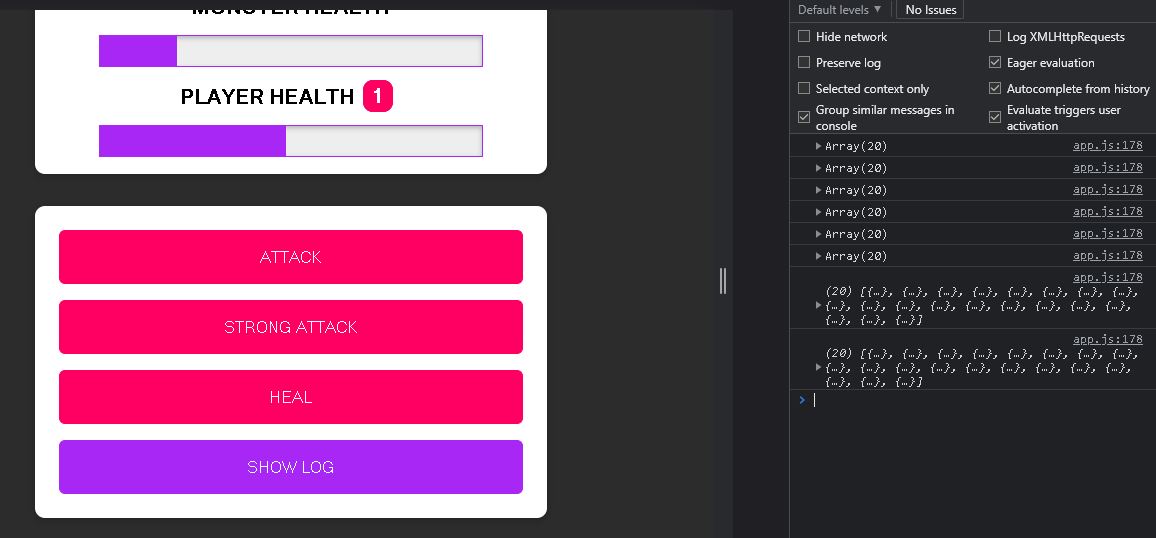


**Validating User Input**

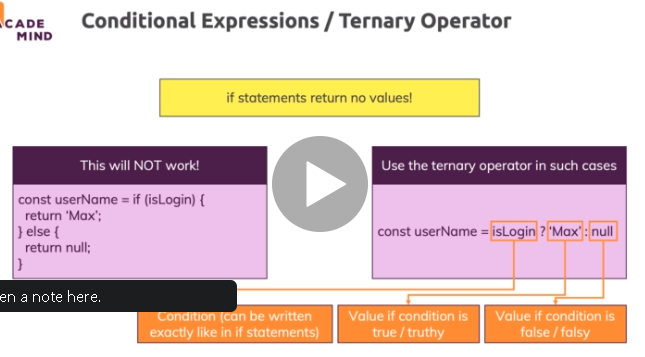


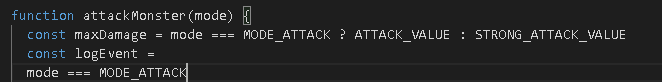


**Adding a Conditional Battle Log**

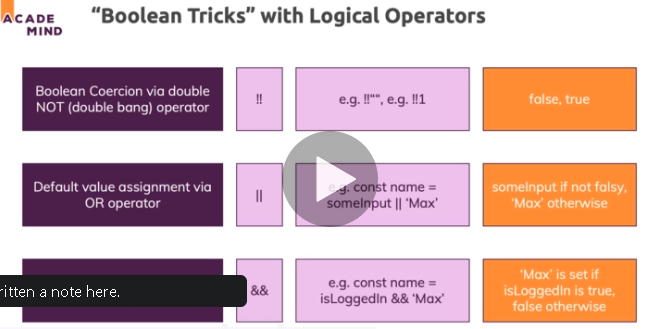


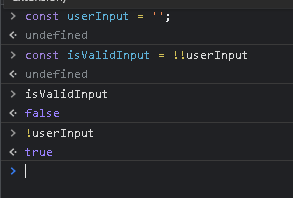
**Introducing the Ternary Operator**





**Logical Operator "Tricks" & Shorthands**





**Working with the "switch-case" Statement**

var age = prompt("Enter you age");

switch (age) {

    case <13:

        alert("You must be 13 or older to play");

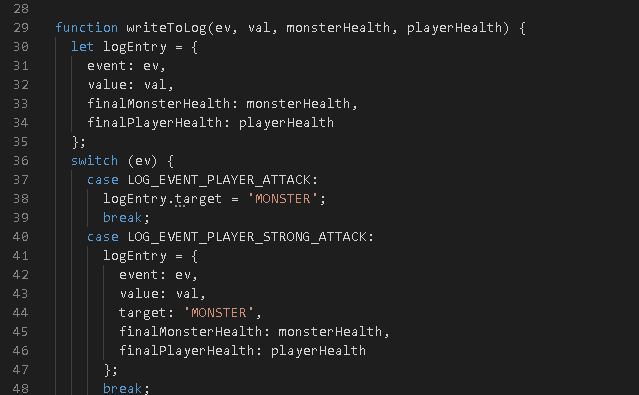
        break;

    case >=13:

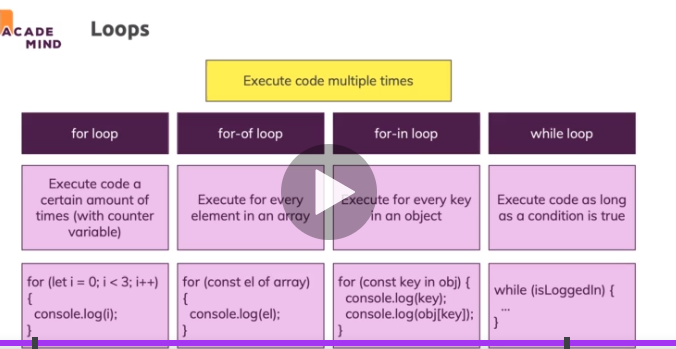
        alert("You are old enough to play");

        break;

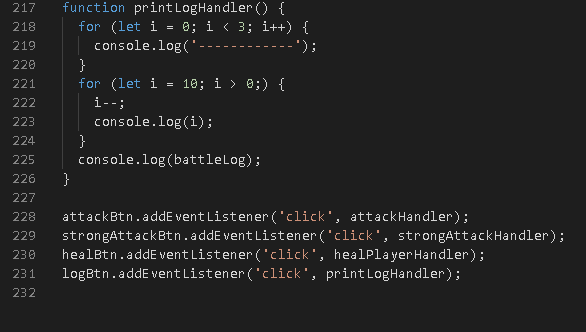
}

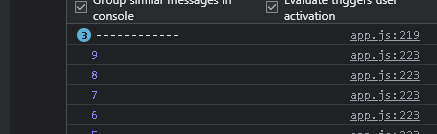


**Introducing Loops**

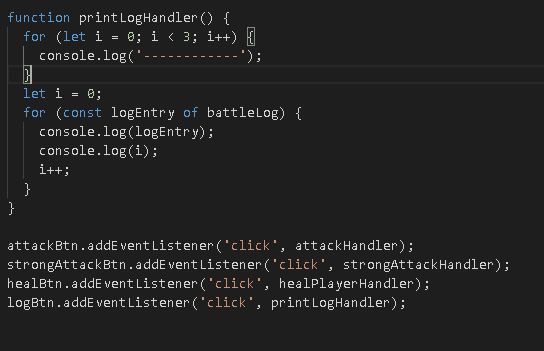


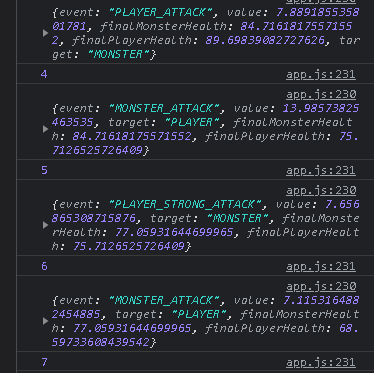
**The "for" Loop**



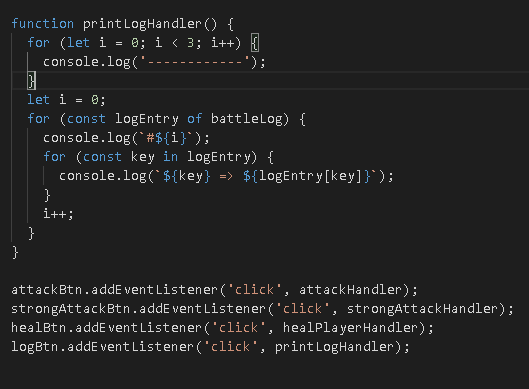


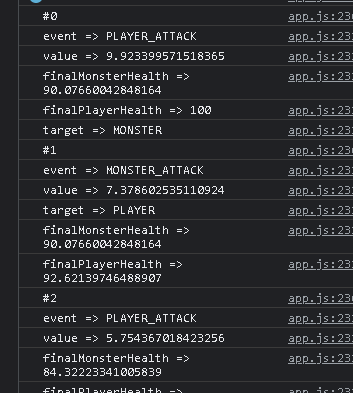
**The "for-of" Loop**



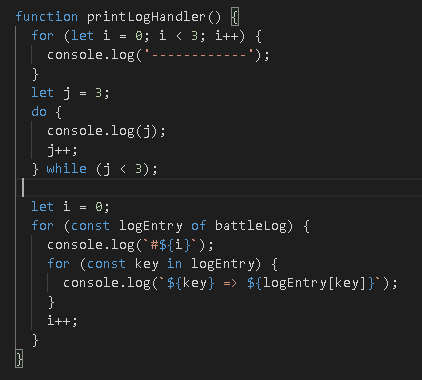


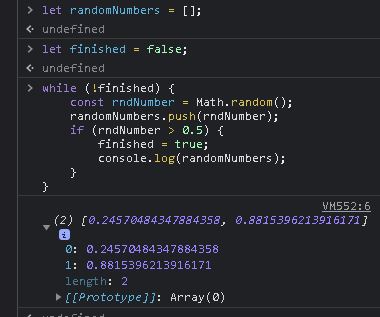
**The "for-in" Loop**



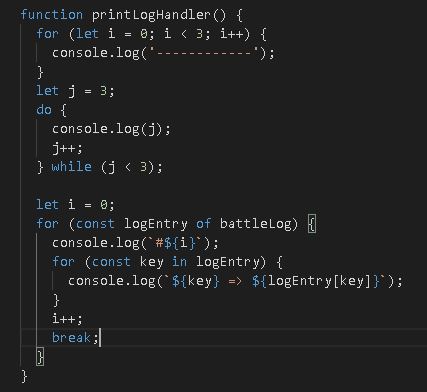


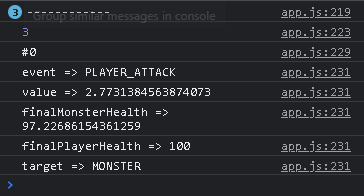
**The "while" & "do-while" Loops**





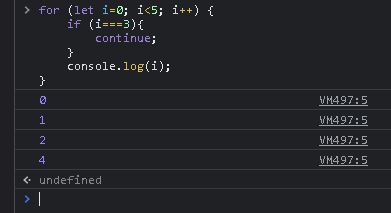
**Controlling Loops with "break"**





**Controlling Iterations with "continue"**





**Error handling**

