Conditional and Logical Operator

Ternary Operator

```
let accessAllowed;

if (age > 18) {
  accessedAllowed = true;
}
else {
  accessedAllowed = false;
}

// Can be simplified into just
accessedAllowed = age > 18 ? true : false;
```

|| (OR)

Returns true if one of the boolean value is true.

There is an "extra" feature of JavaScript for the OR operator.

```
result = value1 || value2 || value3;
```

Given the above code snippet, the OR operator will go from left to right evaluating each expressions, and for each value it will convert it into a boolean, if the result is `true` it will stop and return the original value of that expression. if all of the expressions has been evaluated, and all of them are `false` then return the last value.

Return the first truthy value or the last one if none were found.

```
let firstName = "";
let lastName = "";
let nickName = "SuperCoder";

alert( firstName || lastName || nickName || "Anonymous"); // SuperCoder
```

&& (AND)

Return true only if both operand are true.

Just like OR there is also this "extra" feature from JavaScript for AND operator.

```
result = value1 && value2 && value3;
```

It will evaluate left to right as well. It will convert each value to a boolean, if the result is `false` then it will stop and return the original value. If all of the values are evaluated and are all `true`, then return the last value.

Return the first falsey value or the last one if none were found.

```
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5
```

! (NOT)

It first convert the operand to a boolean type, then return the inverse of that value.

```
result = !value;
```

Double NOT can be used to convert value to boolean conversion.

Nullish coalescing ??

A value is defined when it's neither `null` or `undefined`.

The result of `a ?? b` is

- if `a` is defined, then `a`
- if `a` isn't defined, then `b`

Basically the `??` operator will return the first argument if it's not `null/undefined`, otherwise, the second one.

This is just a shorter syntax for writing

```
result = (a !== null && a !== undefined) ? a : b;


// Is the same as
```

```
result = a ?? b;
```

You can also use `??` to pick the first value that isn't `null/undefined`.

As opposed to `||`, it returns the first truthy value! This results in a subtle difference:

```
let height = 0;


alert(height || 100); // 100
alert(height ?? 100); // 0
```

We might only want to use a default value (in this case is 100), when the variable is `null/undefined` . However, if you use `||` you will be picking the first truthy value, and since `0` is considered falsey you will be skipping the default value, as opposed to the `??` operator, which does what we want. Only use the 100 default value, if it is `null/undefined`.

Revision #2
Created 2022-12-17 20:58:04 UTC by Tamarine
Updated 2022-12-18 16:14:31 UTC by Tamarine