M02 - JavaScript Fundamentals

•••

Operators

Operators

1. Definition
2. Arithmetic
3. Concatenation
4. Assignment
5. Rest and Exponentiation
6. Increment/Decrement
7. Comparison
8. Logic

Operators

1. Definition

Operators

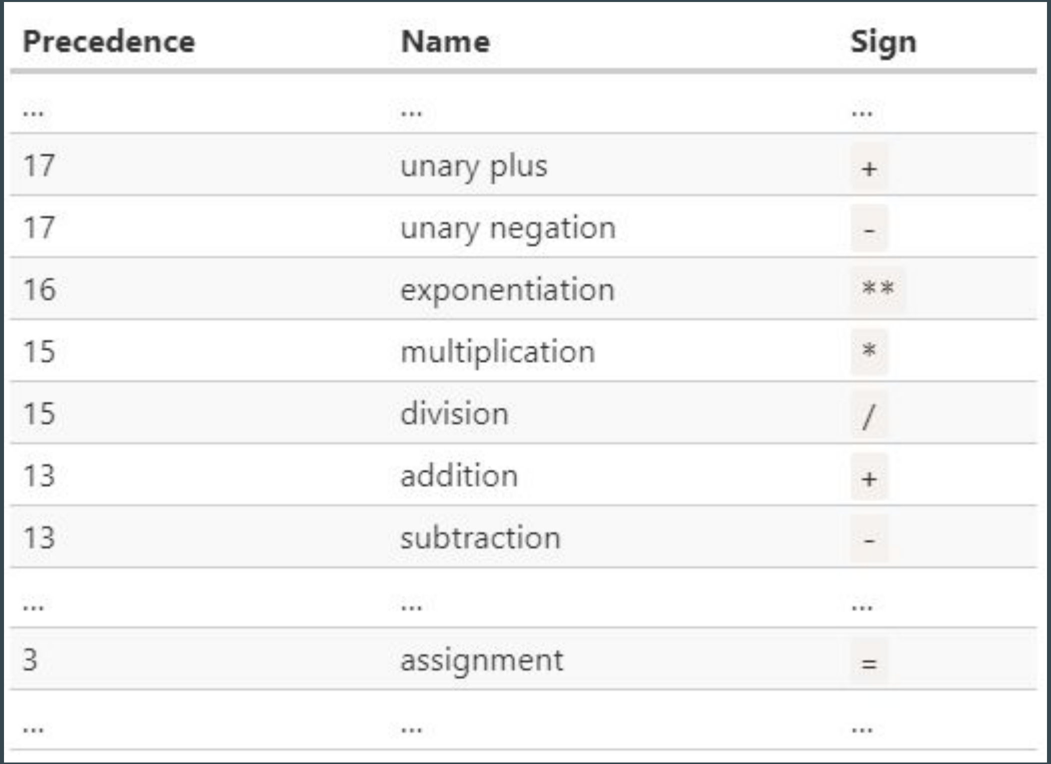
1. Definition

- There are several types of operators in JavaScript
- Types of operators:
  - Arithmetic (+, -, *, /)
  - Concatenation (+)
  - Assignment (=)
  - Rest (%)
  - Exponentiation (**)
  - Increment/Decrement (++, --)
  - Comparison (<, >, <=, >=, ==, !=, ===, !==)
  - Logic (||, && e !)

Operators

1. Definition

- Operator precedence



Precedence	Name	Sign
...	...	...
17	unary plus	+
17	unary negation	-
16	exponentiation	**
15	multiplication	*
15	division	/
13	addition	+
13	subtraction	-
...	...	...
3	assignment	=
...	...	...

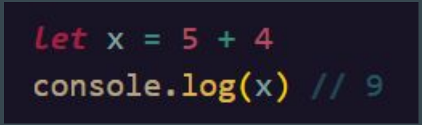
Operators

2. Arithmetic

Operators

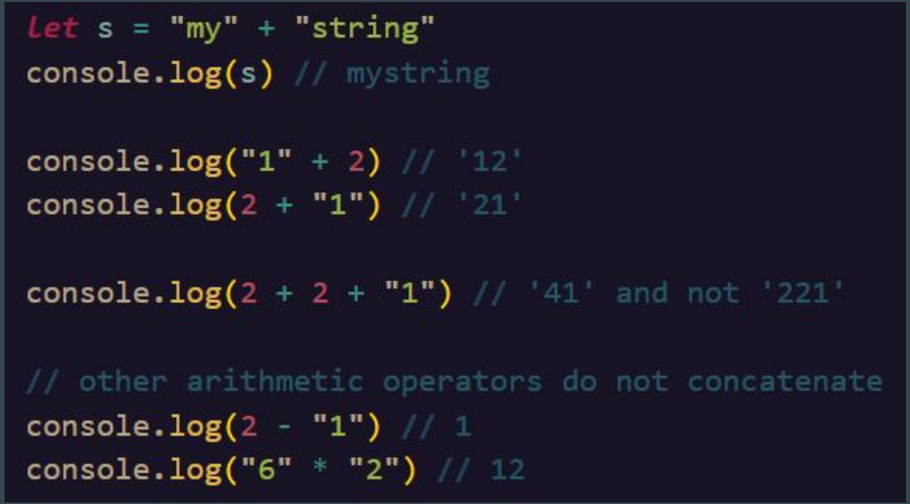
2. Arithmetic Operators (+, -, *, /)

- Used to perform arithmetic operations on numbers (literals or variables)
- Examples of basic operations: +, -, *, /



```
let x = 5 + 4
console.log(x) // 9
```

- The + operator if applied to strings it concatenates them
- If one of the operands is a string, the other will also be converted to a string



```
let s = "my" + "string"
console.log(s) // mystring

console.log("1" + 2) // '12'
console.log(2 + "1") // '21'

console.log(2 + 2 + "1") // '41' and not '221'

// other arithmetic operators do not concatenate
console.log(2 - "1") // 1
console.log("6" * "2") // 12
```

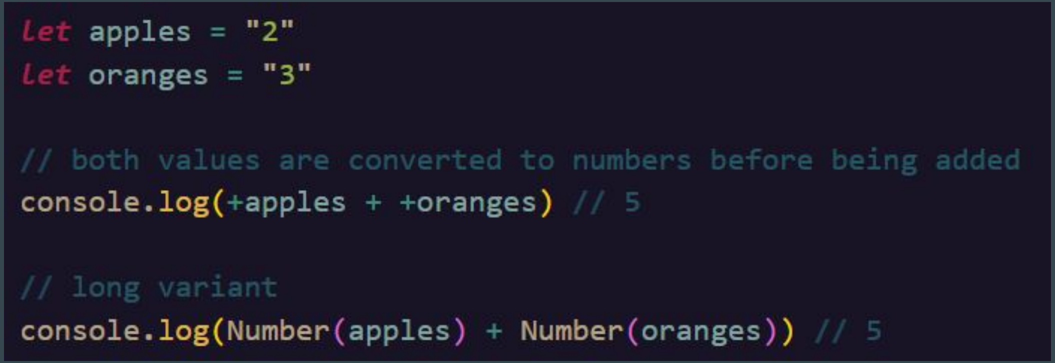
Operators

3. Concatenation

Operators

3. Concatenation Operators (+)

- The + operator if applied to a single value does nothing with numbers
- But if the operand is not a number, the unary + converts it to a number



```
let apples = "2"
let oranges = "3"

// both values are converted to numbers before being added
console.log(+apples + +oranges) // 5

// long variant
console.log(Number(apples) + Number(oranges)) // 5
```

- An operator is unary if it has a single operand
- For example, unary negation - inverses the sign of a number

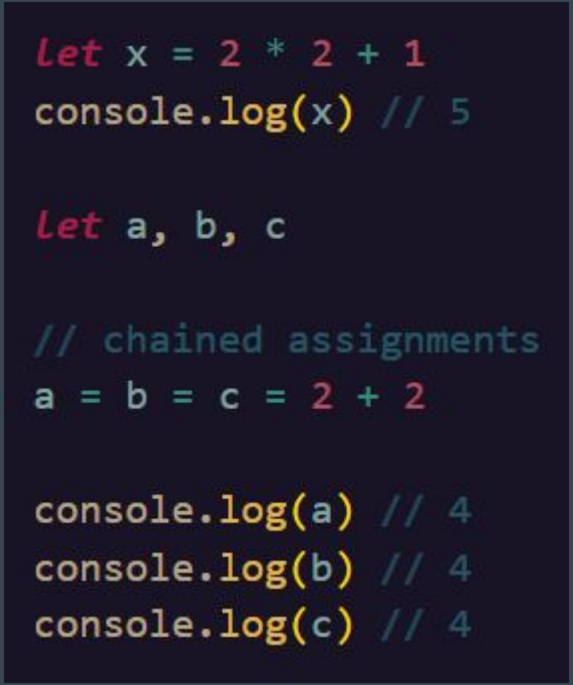
Operators

4. Assignment

Operators

4. Assignment Operators (=)

- The assignment = is also an operator
- It is listed in the precedence table with a very low priority of 3
- That is why, when we assign a variable, such as x = 2 * 2 + 1, calculations are done first and then the = is evaluated, storing the result in x
- It is also possible to chain assignments



```
let x = 2 * 2 + 1
console.log(x) // 5

let a, b, c

// chained assignments
a = b = c = 2 + 2

console.log(a) // 4
console.log(b) // 4
console.log(c) // 4
```

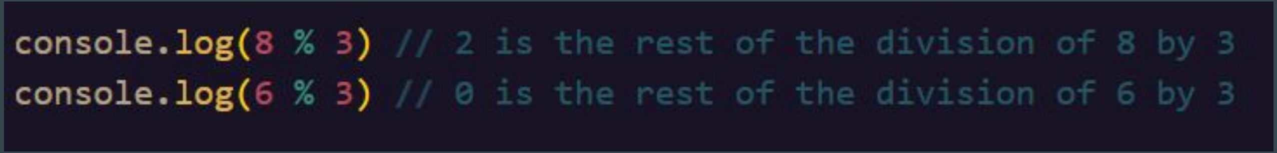
Operators

5. Rest and Exponentiation

Operators

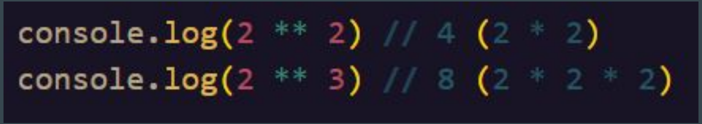
5. Rest (%) and Exponentiation (**) Operators

- Rest
  - The remainder operator (%), despite its appearance, is not related to percentages
  - The result of a%b is the remainder of the entire division of a by b



```
console.log(8 % 3) // 2 is the rest of the division of 8 by 3
console.log(6 % 3) // 0 is the rest of the division of 6 by 3
```

- Exponentiation
  - The exponentiation operator ** is a recent addition to the language
  - A natural number b, the result of a**b is a multiplied by itself b times



```
console.log(2 ** 2) // 4 (2 * 2)
console.log(2 ** 3) // 8 (2 * 2 * 2)
```

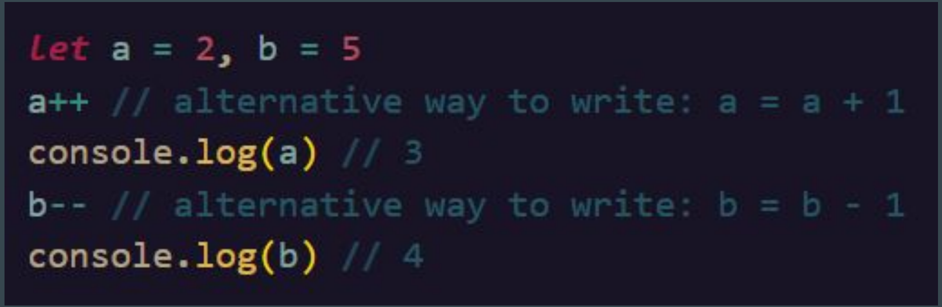
Operators

6. Increment/Decrement

Operators

6. Increment (++) and Decrement (--) Operators

- Increasing or decreasing a number by one is among the most common numerical operations
- There are special operators for this:
  - Increment ++ increases a variable by 1
  - Decrement -- decrease a variable by 1

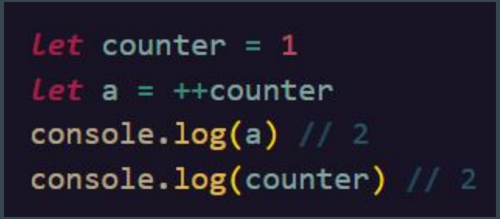


```
let a = 2, b = 5
a++ // alternative way to write: a = a + 1
console.log(a) // 3
b-- // alternative way to write: b = b - 1
console.log(b) // 4
```

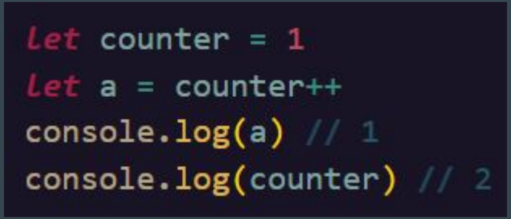
Operators

6. Increment (++) and Decrement (--) Operators

- ++ and -- operators can be placed before or after a variable
  - When the operator is after the variable, it is in the postfix form: counter ++
  - When the operator is before the variable, it is in the prefix form: ++counter
- Both statements do the same thing: increase the counter by 1
- So what's the difference?
  - The prefix returns the new value and postfix returns the old value (before the increment/decrement)



```
let counter = 1
let a = ++counter
console.log(a) // 2
console.log(counter) // 2
```

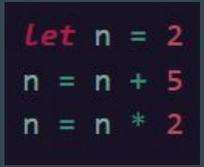


```
let counter = 1
let a = counter++
console.log(a) // 1
console.log(counter) // 2
```

Operators

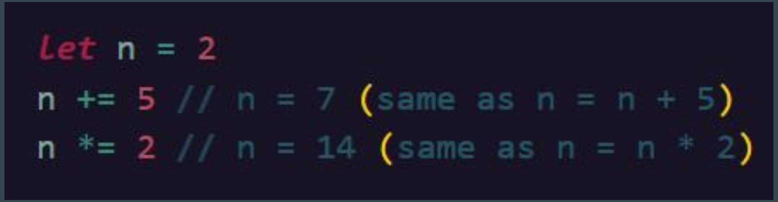
6. Increment (++) and Decrement (--) Operators

- Generally, we need to apply an operator to a variable and store the new result in that same variable



```
let n = 2
n = n + 5
n = n * 2
```

- This notation can be shortened using the operators += and *=



```
let n = 2
n += 5 // n = 7 (same as n = n + 5)
n *= 2 // n = 14 (same as n = n * 2)
```

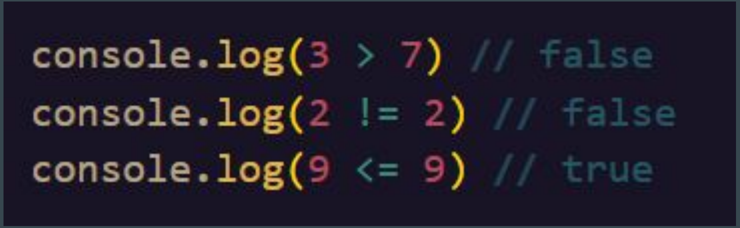
Operators

7. Comparison

Operators

7. Comparison Operators (>, <, >=, <=, ==, !=, ===, !==)

- We know many math comparison operators:
  - Greater/less than: a > b, a < b
  - Greater/less than or equal to: a >= b, a <= b
  - Equal: a == b (note the double equal sign =. A single symbol a = b would mean an assignment)
  - Not equal. In mathematics, notation is ≠, but in JavaScript it is written as an assignment with an exclamation sign before it: a != b
- A comparison returns a Boolean value

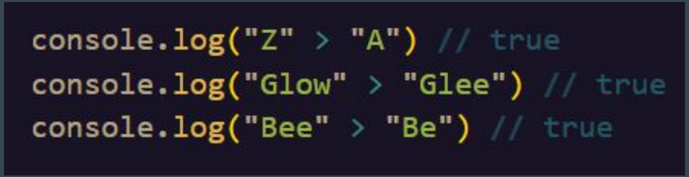


```
console.log(3 > 7) // false
console.log(2 != 2) // false
console.log(9 <= 9) // true
```

Operators

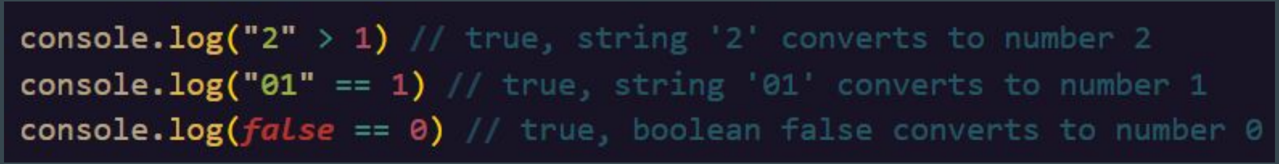
7. Comparison Operators (>, <, >=, <=, ==, !=, ===, !==)

- String comparison
  - To see if a string is longer than another, JS uses the so-called dictionary or lexicographic
  - In other words, strings are compared letter by letter



```javascript
console.log("Z" > "A") // true
console.log("Glow" > "Glee") // true
console.log("Bee" > "Be") // true
```

- Comparison of different types
  - When comparing values of different types, JavaScript converts values into numbers

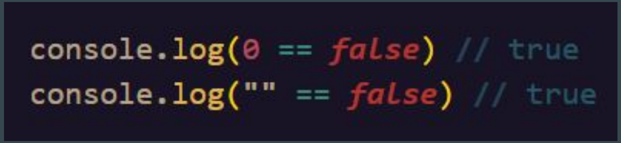


```javascript
console.log("2" > 1) // true, string '2' converts to number 2
console.log("01" == 1) // true, string '01' converts to number 1
console.log(false == 0) // true, boolean false converts to number 0
```

Operators

7. Comparison Operators (>, <, >=, <=, ==, !=, ===, !==)

- Strict equality
  - A regular equality check == has a problem
  - It is not possible to differentiate, for example, 0 from false:



```
console.log(0 == false) // true
console.log("" == false) // true
```

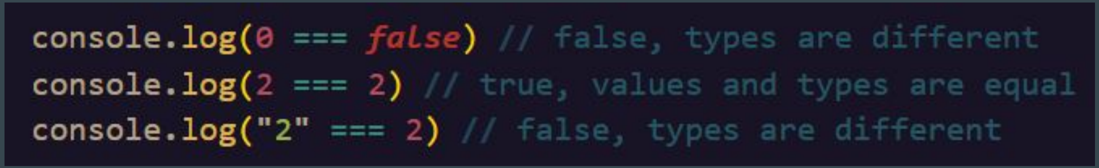
  - This is because operands of different types are converted to numbers by the equality operator ==. An empty string, as a false, becomes a zero.

  - Here comes the strict equality operator === which checks for equality without type conversion

Operators

7. Comparison Operators (>, <, >=, <=, ==, !=, ===, !==)

- Strict equality
  - A strict equality operator === checks for equality without type conversion.
  - In other words, if a and b are of different types, then a === b immediately returns false without an attempt to convert them.



```
console.log(0 === false) // false, types are different
console.log(2 === 2) // true, values and types are equal
console.log("2" === 2) // false, types are different
```

  - There is also a strict non-egalitarian operator !== analogous to !=

Operators

8. Logic

Operators

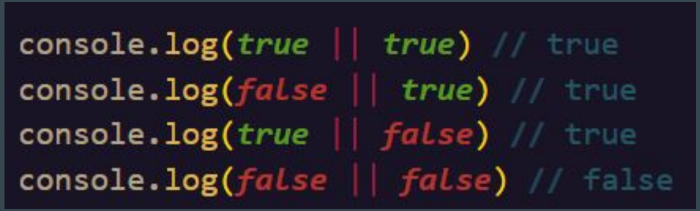
8. Logical Operators (||, && e !)

- There are three logical operators in JavaScript:
  - || (OR)
  - && (AND)
  - ! (NOT)
- Although they are called logical, they can be applied to values of any type, not just Booleans.
- Their result can also be of any kind

Operators

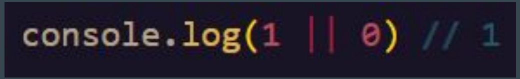
8. Logical Operators (||, && e !)

- Operator || (OR)
  - In classical programming, the logical OR is intended to manipulate only Boolean values
  - If any of its arguments are true, it will return true, otherwise it will return false



```
console.log(true || true) // true
console.log(false || true) // true
console.log(true || false) // true
console.log(false || false) // false
```

  - If an operand is not Boolean, it will be converted to Boolean for evaluation
  - For example, the number 1 is treated as true, the number 0 as false:

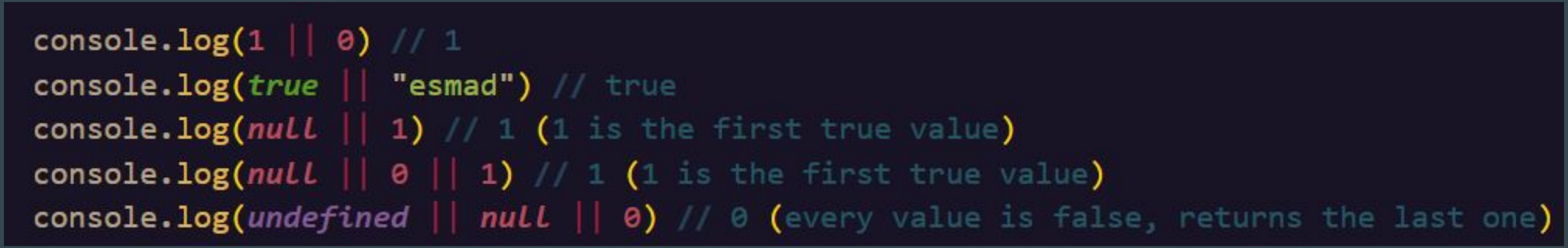


```
console.log(1 || 0) // 1
```

Operators

8. Logical Operators (||, && e !)

- Operator || (OR)
  - Given various OR values: result = value1 || value2 || value3;
  - The operator || does the following:
    - Evaluates operands from left to right
    - For each operand, convert it to Boolean. If the result is true, it will stop and return the original value of that operand.
    - If all operands have been evaluated (all false), returns the last operand.
  - A value is returned in its original form, without conversion.

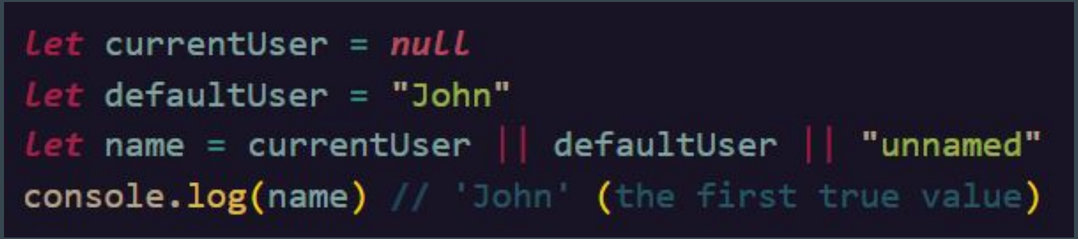


```
console.log(1 || 0) // 1
console.log(true || "esmad") // true
console.log(null || 1) // 1 (1 is the first true value)
console.log(null || 0 || 1) // 1 (1 is the first true value)
console.log(undefined || null || 0) // 0 (every value is false, returns the last one)
```

Operators

8. Logical Operators (||, && e !)

- Operator || (OR)
  - Useful for obtaining the first true value from a list of variables or expressions
  - Imagine that we have several variables that can contain data or be null/undefined. How can we find the first one with data?

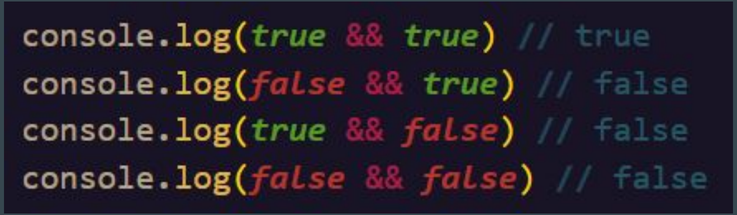


```
let currentUser = null
let defaultUser = "John"
let name = currentUser || defaultUser || "unnamed"
console.log(name) // 'John' (the first true value)
```

Operators

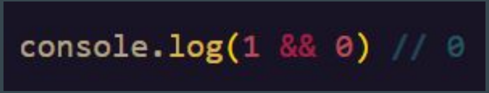
8. Logical Operators (||, && e !)

- Operator && (AND)
  - The AND operator returns true if both operands are true and false otherwise



```
console.log(true && true) // true
console.log(false && true) // false
console.log(true && false) // false
console.log(false && false) // false
```

  - If an operand is not Boolean, it will be converted to Boolean for evaluation
  - For example, the number 1 is treated as true, the number 0 as false:

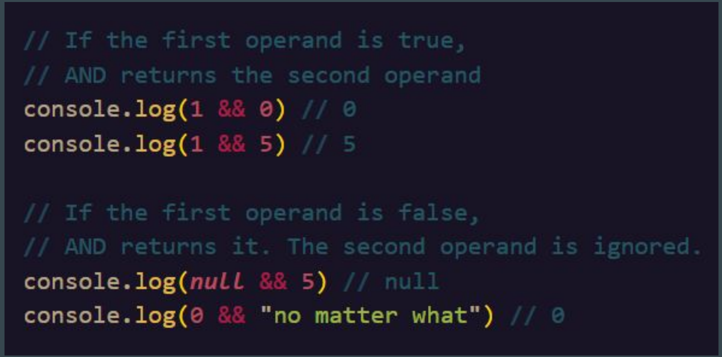


```
console.log(1 && 0) // 0
```

Operators

8. Logical Operators (||, && e !)

- Operator && (AND)
  - Several values: result = value1 && value2 && value3;
  - The && operator does the following:
    - Evaluates operands from left to right
    - For each operand, convert it to a Boolean. If the result is false, it will stop and return the original value of that operand.
    - If all operands have been evaluated (that is, they were all true), the last operand will be returned.
  - In other words, the && operator returns the first false value or the last value, if they are all true.



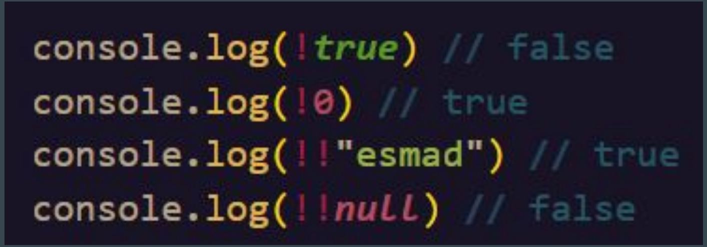
```
// If the first operand is true,
// AND returns the second operand
console.log(1 && 0) // 0
console.log(1 && 5) // 5

// If the first operand is false,
// AND returns it. The second operand is ignored.
console.log(null && 5) // null
console.log(0 && "no matter what") // 0
```

Operators

8. Logical Operators (||, && e !)

- Operator ! (NOT)
  - The operator accepts a single argument and does the following:
    - Converts the operand to a Boolean type: true/false
    - Returns the inverse value
    - A double !! is used to convert a value to a Boolean type



```
console.log(!true) // false
console.log(!0) // true
console.log(!!"esmad") // true
console.log(!!null) // false
```