2b Theory JS API pages 23 - 38 Strings & Operators

Nicholas Bruzzese

July 23, 2025

Introduction

This document explains three key concepts in JavaScript: **strings**, **Booleans**, and **operators**. These are like the building blocks for your coding projects.

1 Strings

Strings are like words or sentences in JavaScript. They're made up of characters, which can be letters, numbers, symbols, or spaces—basically anything you can type. To tell JavaScript it's a string, you wrap it in quotation marks, like `"hello"` or `"123"`. You can use double quotes (`"`) or single quotes (`'`), but we'll stick with double quotes here for simplicity.

For example:

```
"hello world" // This is a string
"123" // This is a string too, even though it looks like a number
```

You can store strings in variables, like putting a note in a labelled box to use later:

```
var myString = "Something REALLY awesome!";
console.log(myString); // Prints: Something REALLY awesome!
```

1.1 Joining Strings

You can combine strings using the `+` operator, which sticks one string to the end of another. This is called *concatenation*. Think of it like sticking two bits of paper together to make a longer one:

```
var greeting = "G'day";
var myName = "Nick";
var fullGreeting = greeting + myName; // Makes "G'dayNick"
```

Notice there's no space between `G'day` and `Nick`. If you want a space, you have to add it yourself:

```
var greeting = "G'day ";
var myName = "Nick";
var fullGreeting = greeting + myName; // Makes "G'day Nick"
```

1.2 Finding the Length of a String

To find out how many characters are in a string, use `.length`. It's like counting the letters in a word, including spaces and punctuation:

```
"supercalifragilisticexpialidocious".length; // Returns 34
var java = "Java";
java.length; // Returns 4
```

This works whether you use it on a string directly or on a variable holding a string.

1.3 Getting a Single Character

To grab just one character from a string, use square brackets `[ ]` with the position number of the character. JavaScript starts counting at 0, so the first character is at position 0, the second at 1, and so on. It's like picking a specific letter from a word:

```
var myName = "Nick";
myName[0]; // Returns "N"
myName[1]; // Returns "i"
```

For example, if you want to decode a secret message by taking the second character from a few words:

```
var word1 = "are";
var word2 = "tubas";
var word3 = "unsafe";
var word4 = "list";
var secret = word1[1] + word2[1] + word3[1] + word4[1]; //
   Returns "runt"
```

1.4 Cutting Up Strings

Sometimes you want just a piece of a string, like cutting a slice of cake. The `.slice()` method lets you do this by specifying a start position and an end position (the character *after* the last one you want):

```
var longString = "My long string is long";
longString.slice(3, 14); // Returns "long string"
```

If you only give one number, `.slice()` takes everything from that position to the end:

```
longString.slice(3); // Returns "long string is long"
```

1.5 Changing Case

You can make a string all uppercase (like shouting) or all lowercase (like speaking quietly) using `.toUpperCase()` and `.toLowerCase()`:

```
    "G'day there".toUpperCase(); // Returns "G'DAY THERE"
    "G'DAY THERE".toLowerCase(); // Returns "g'day there"
```

If you want just the first letter capitalised (like in a proper sentence), you can combine these with `.slice()`:

```
    var sillyString = "hELLO MATE, HOW ARE YOU?";
    var lowerString = sillyString.toLowerCase(); // "hello mate,
       how are you?"
    var firstChar = lowerString[0].toUpperCase(); // "H"
    var restOfString = lowerString.slice(1); // "ello mate, how
       are you?"
    var fixedString = firstChar + restOfString; // "Hello mate,
       how are you?"
```

This is like tidying up a messy sentence to make it look proper.

2 Booleans

A Boolean is a value that's either `true` or `false`. Think of it like a light switch: it's either on (`true`) or off (`false`). Booleans are great for answering yes-or-no questions in your code:

```
    var javascriptIsCool = true;
    console.log(javascriptIsCool); // Prints: true
```

2.1 Logical Operators

You can combine Booleans using special operators to make decisions, like figuring out if you're ready for school. There are three main logical operators:

1. **`&&` (AND)**
   This means "and". It's `true` only if *both* things are true. Imagine you need to have your school bag *and* have showered to go to school:

   ```
       var hasShower = true;
       var hasSchoolBag = false;
       hasShower && hasSchoolBag; // Returns false (because
          hasSchoolBag is false)
   ```

2. **`||` (OR)**
   This means "or". It's `true` if *at least one* thing is true. If you have an apple *or* a banana, you have fruit:

```
    var hasApple = false;
    var hasBanana = true;
    hasApple || hasBanana; // Returns true (because
       hasBanana is true)
```

3. **`!` (NOT)**
   This means "not" and flips a Boolean. If something is `true`, `!` makes it `false`, and vice versa. It's like saying the opposite:

```
    var isWeekend = true;
    var needToShower = !isWeekend; // Returns false
       (because it's the weekend)
```

2.2 Combining Logical Operators

You can mix these operators to make complex decisions. For example, you go to school if it's *not* the weekend, you've showered, *and* you have either an apple *or* a banana:

```
    var isWeekend = false;
    var hadShower = true;
    var hasApple = false;
    var hasBanana = true;
    var shouldGoToSchool = !isWeekend && hadShower && (hasApple
       || hasBanana);
    console.log(shouldGoToSchool); // Returns true
```

The parentheses around `(hasApple || hasBanana)` tell JavaScript to check that part first, just like in maths where you do what's in brackets first.

3 Operators for Comparing Numbers

You can use Booleans to compare numbers, which is useful for things like checking if someone is tall enough for a ride at a theme park. Here are the operators:

1. **`>` (Greater Than)**
   Checks if one number is bigger than another:

```
    var height = 165;
    var heightRestriction = 150;
    height > heightRestriction; // Returns true (165 is
       greater than 150)
```

2. **`>=` (Greater Than or Equal To)**
   Checks if a number is bigger than or equal to another. This is great for things like a height restriction where "exactly 150 cm" is okay:

```
var height = 150;
var heightRestriction = 150;
height >= heightRestriction; // Returns true (150 is
   equal to 150)
```

3. **< (Less Than)**
   Checks if a number is smaller than another. For a kids-only ride:

```
var height = 150;
var heightRestriction = 120;
height < heightRestriction; // Returns false (150 is
   not less than 120)
```

4. **<= (Less Than or Equal To)**
   Checks if a number is smaller than or equal to another:

```
var height = 120;
var heightRestriction = 120;
height <= heightRestriction; // Returns true (120 is
   equal to 120)
```

5. **=== (Equal To)**
   Checks if two values are exactly the same, including their type (number, string, etc.):

```
var mySecretNumber = 5;
var guess = 5;
mySecretNumber === guess; // Returns true
```

   But:

```
var stringNumber = "5";
var actualNumber = 5;
stringNumber === actualNumber; // Returns false
   (different types)
```

6. **== (Double Equals)**
   This is a looser version of ===. It checks if two values are "equal-ish", even if they're different types. JavaScript tries to convert them to match:

```
var stringNumber = "5";
var actualNumber = 5;
stringNumber == actualNumber; // Returns true
   (JavaScript converts "5" to 5)
```

   Be careful with == because it can be confusing. For example:

```
0 == false; // Returns true (0 is considered
   "false-ish")
"false" == false; // Returns false (the string "false"
   isn't the same as false)
```

   Stick with === unless you really need ==, because === is clearer and avoids surprises.

4 Undefined and Null

These are special values in JavaScript that mean "nothing", but they're used differently:

1. **Undefined**
   This is what JavaScript gives you when a variable exists but hasn't been given a value. It's like an empty box you haven't filled yet:

   ```
       var myVariable;
       console.log(myVariable); // Returns undefined
   ```

2. **Null**
   This is when you *choose* to say something is empty. It's like labelling a box "empty" on purpose:

   ```
       var myNullVariable = null;
       console.log(myNullVariable); // Returns null
   ```

   Use `null` when you want to clearly say "this has no value". For example, if you don't have a favourite veggie, you might set:

   ```
       var favouriteVeggie = null;
   ```

   If it's `undefined`, someone might think you just forgot to set it.

5 Putting It All Together: Cinema Example

The document gives an example where you need to decide if a 12-year-old can see a PG-13 movie. The rule is: they must be 13 or older, *or* they must be accompanied by an adult. Here's how you'd write it:

```
    var age = 12;
    var accompanied = true;
    var canSeeMovie = age >= 13 || accompanied;
    console.log(canSeeMovie); // Returns true (because
       accompanied is true)
```

If you change the values:

```
    var age = 13;
    var accompanied = false;
    var canSeeMovie = age >= 13 || accompanied;
    console.log(canSeeMovie); // Returns true (because age is 13)
```

But if they're under 13 and not accompanied:

```
    var age = 12;
    var accompanied = false;
    var canSeeMovie = age >= 13 || accompanied;
    console.log(canSeeMovie); // Returns false
```

This uses the `>=` operator to check age and the `||` operator to combine conditions.

6 Summary

- **Strings** are like words or sentences in quotation marks. You can join them with `+`, find their length with `.length`, grab single characters with `[ ]`, cut pieces with `.slice()`, and change their case with `.toUpperCase()` or `.toLowerCase()`.
- **Booleans** are `true` or `false` values, like yes-or-no answers. You combine them with `&&` (and), `||` (or), and `!` (not) to make decisions.
- **Operators** let you compare numbers (`>`, `>=`, `<`, `<=`, `===`, `==`) to get Boolean results, which help you make choices in code.
- **Undefined** means a variable has no value yet, while `null` means you've chosen to say it's empty.

These tools are like building blocks for your JavaScript programs. You can use them to create messages, make decisions, and check conditions, like figuring out if someone can go on a ride or watch a movie at the cinema. Keep practising with small examples, and you'll get the hang of how they work together!