Javascript Day - 3

Type Coercion, String Methods

Type Coercion

JavaScript is a **dynamically typed language**, meaning that **variables can hold values of any data type without explicit declaration.**

1. Implicit Coercion
2. Explicit Coercion

Explicit Coercion

These are performed explicitly by developer using the below

1. Number()
2. String()
3. Boolean()
4. parseInt()
5. parseFloat()
6. toString()

Implicit Coercion

Occurs automatically by JavaScript during operations.

JavaScript attempts to convert the operands of an operation to a common data type when they have different types. This happens automatically during certain operations such as arithmetic operations, comparisons, and concatenations.

Coercion Rules

1. String
2. Number
3. Boolean

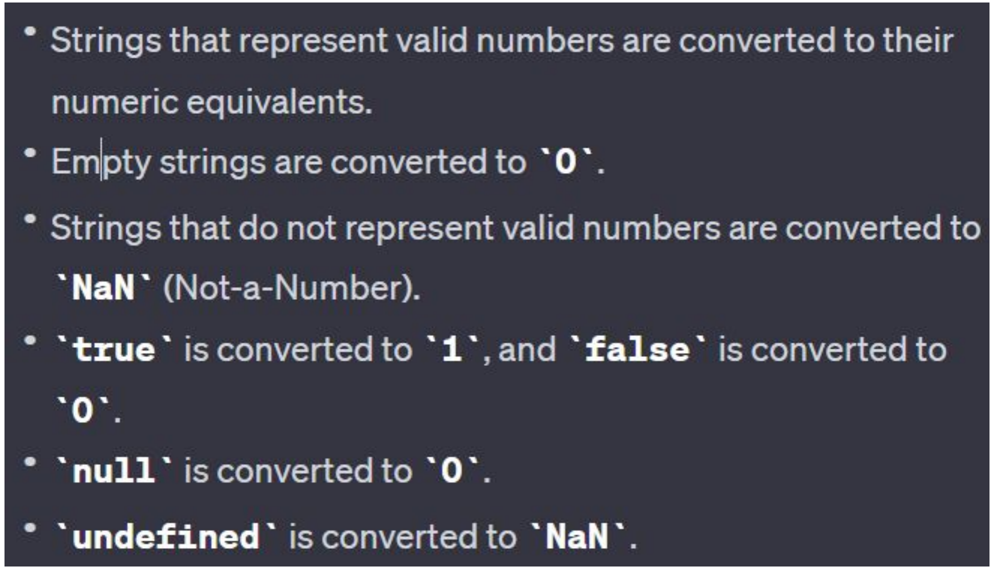
Coercion Rules - String

When a non-string value is used in a string context (**such as concatenation with a string**), JavaScript converts the value to a string

Example: Concatenation with + operator

Coercion Rules - Number

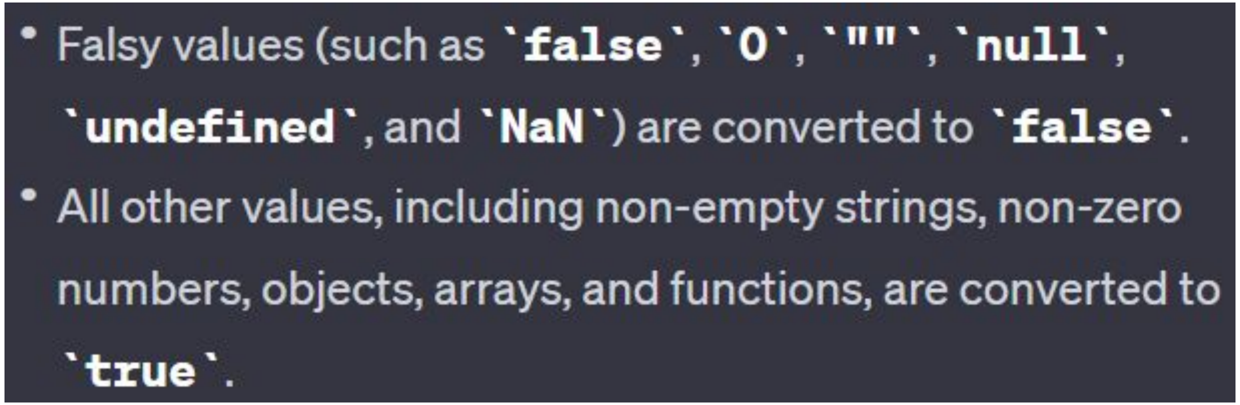
When a non-numeric value is used in a numeric context (**such as arithmetic operations**), JavaScript converts the value to a number. This conversion follows specific rules:



- Strings that represent valid numbers are converted to their numeric equivalents.
- Empty strings are converted to `0`.
- Strings that do not represent valid numbers are converted to `NaN` (Not-a-Number).
- `true` is converted to `1`, and `false` is converted to `0`.
- `null` is converted to `0`.
- `undefined` is converted to `NaN`.

Coercion Rules - Boolean

When a non-boolean value is used in a boolean context (such as with **logical operators or in control structures**), JavaScript converts the value to a boolean. Most values in JavaScript can be converted to either true or false, following certain rules:

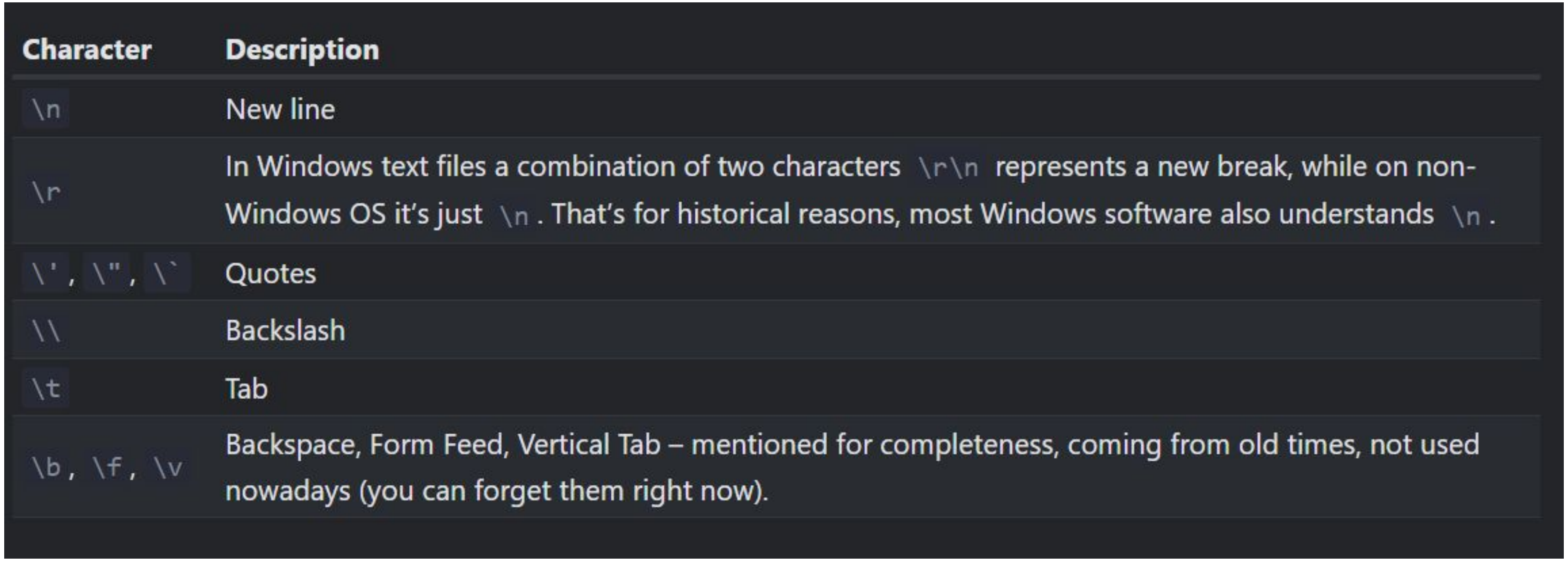


- Falsy values (such as `false`, `0`, `""`, `null`, `undefined`, and `NaN`) are converted to `false`.
- All other values, including non-empty strings, non-zero numbers, objects, arrays, and functions, are converted to `true`.

Strings In Depth

1. Special Characters
2. String Length
3. Accessing Characters
4. Strings are immutable
5. Changing the case
6. Searching for substring
7. Getting a substring
8. Comparing Substring

Strings - Special Characters



Character	Description
`\n`	New line
`\r`	In Windows text files a combination of two characters `\r\n` represents a new break, while on non-Windows OS it's just `\n` . That's for historical reasons, most Windows software also understands `\n` .
`\'` , `\"` , `` \` ``	Quotes
`\\`	Backslash
`\t`	Tab
`\b` , `\f` , `\v`	Backspace, Form Feed, Vertical Tab – mentioned for completeness, coming from old times, not used nowadays (you can forget them right now).

String - length

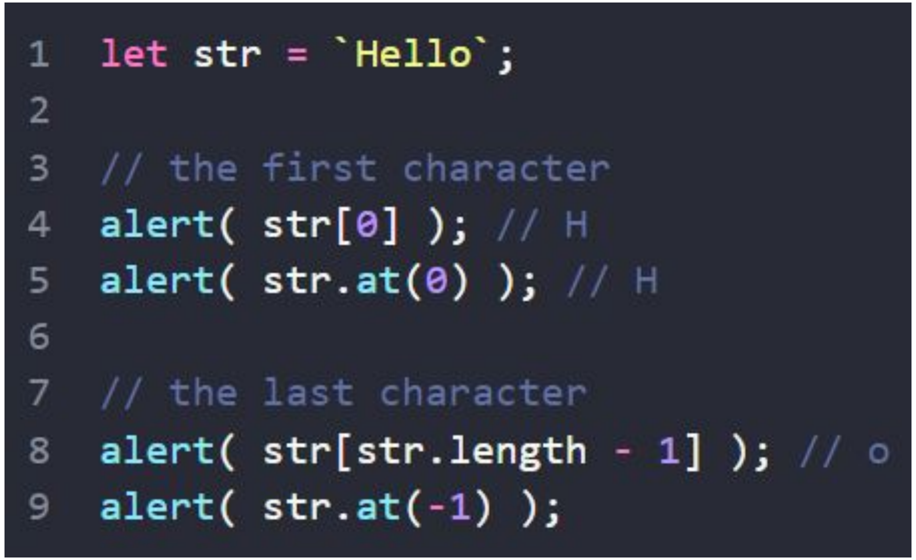
str.length: The length property has the string length

Note: length is a **property** of string. It is not a method or a function. Don't confuse it with **.length()**

⚠️ `length` is a property

String - Accessing Characters

To get a character at position pos, use square brackets [pos] or call the method str.at(pos). The first character starts from the zero position:

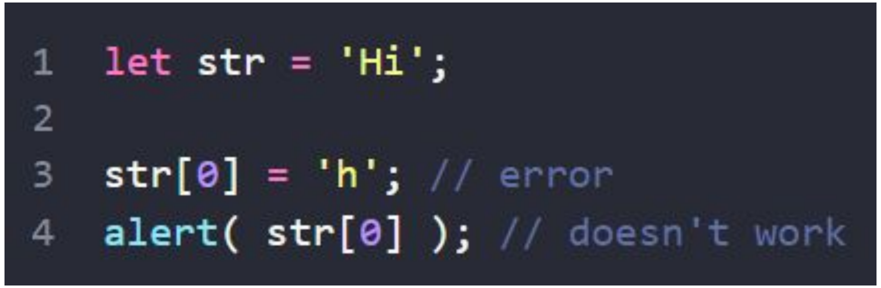


```
let str = `Hello`;

// the first character
alert( str[0] ); // H
alert( str.at(0) ); // H

// the last character
alert( str[str.length - 1] ); // o
alert( str.at(-1) );
```

String - Immutable

Immutable: Cannot be changes

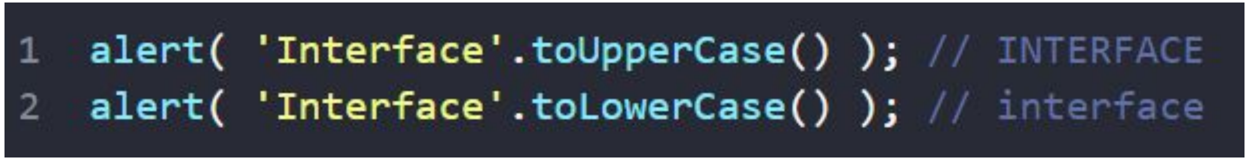
Strings can’t be changed in JavaScript. It is impossible to change a character.



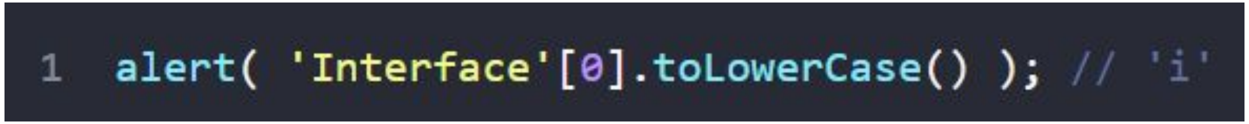
```
let str = 'Hi';

str[0] = 'h'; // error
alert( str[0] ); // doesn't work
```

String - Changing the case

1. toLowerCase() method Changes the case to lowercase.
2. toUpperCase() method changes the case to uppercase.



```
alert( 'Interface'.toUpperCase() ); // INTERFACE
alert( 'Interface'.toLowerCase() ); // interface
```



```
alert( 'Interface'[0].toLowerCase() ); // 'i'
```

String - Searching for substring

Substring: A substring is defined as a contiguous part of a string, i.e., a string inside another string.

Example: str = “Navgurukul”, here **gurukul** is substring of str.

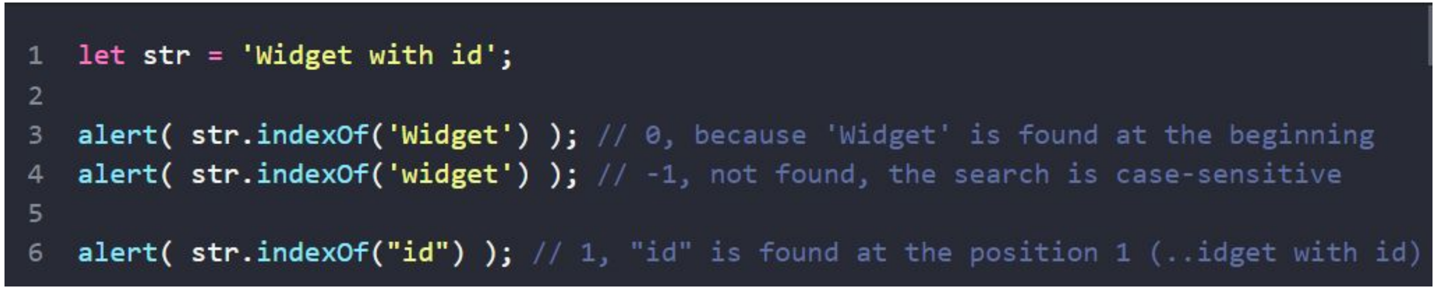
1. str,indexOf() method

2. str.lastIndexOf() method

String - Searching for subtring

1. **str.indexOf(substr, pos)**: It looks for the **substr** in str, starting from the given position **pos**, and returns the position where the match was found or **-1 if nothing can be found**.

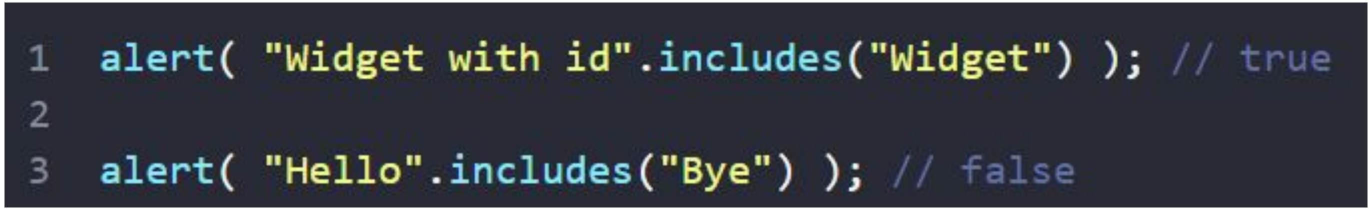
2. **str.lastIndexOf()**: searches from the end of a string to its beginning. It would list the occurrences in the reverse order.



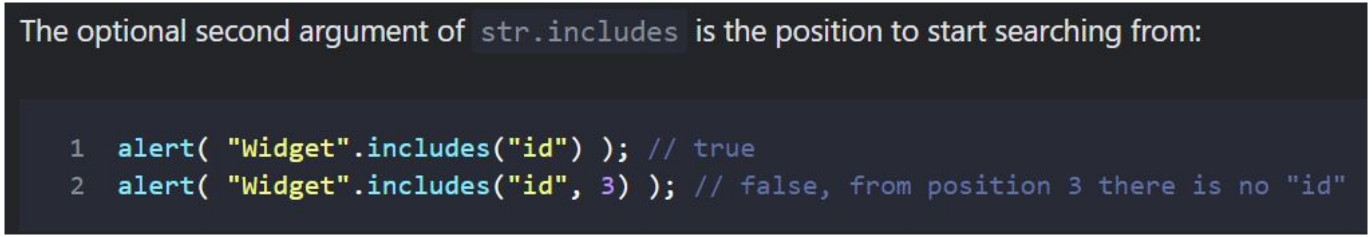
```
let str = 'Widget with id';

alert( str.indexOf('Widget') ); // 0, because 'Widget' is found at the beginning
alert( str.indexOf('widget') ); // -1, not found, the search is case-sensitive

alert( str.indexOf("id") ); // 1, "id" is found at the position 1 (..idget with id)
```

String - Searching a substring

1. includes()
2. startsWith()
3. endsWith()

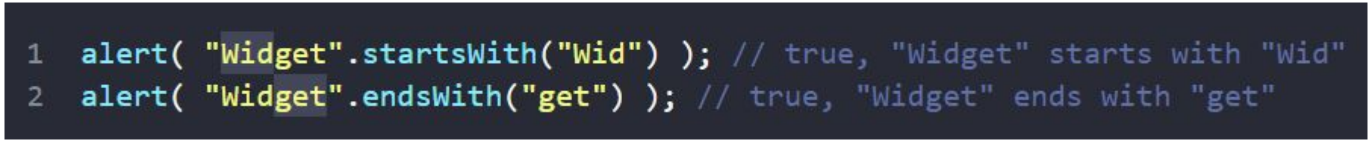


```
alert( "Widget with id".includes("Widget") ); // true

alert( "Hello".includes("Bye") ); // false
```



The optional second argument of `str.includes` is the position to start searching from:

```
alert( "Widget".includes("id") ); // true
alert( "Widget".includes("id", 3) ); // false, from position 3 there is no "id"
```



```
alert( "Widget".startsWith("Wid") ); // true, "Widget" starts with "Wid"
alert( "Widget".endsWith("get") ); // true, "Widget" ends with "get"
```

String - Getting a substring

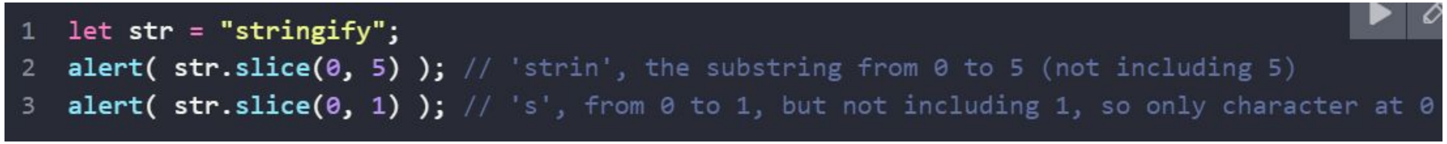
There are 3 methods in JavaScript to get a substring: **substring**, **substr** and **slice**

1. **slice**
2. **substring**
3. **substr**

String - Getting a substring (slice)

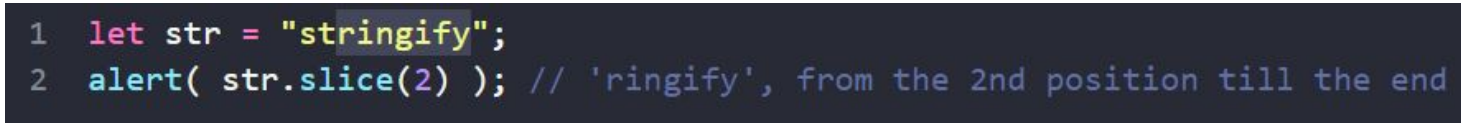
slice: str.slice(start [, end])

Returns the part of the string from start to (but not including) end



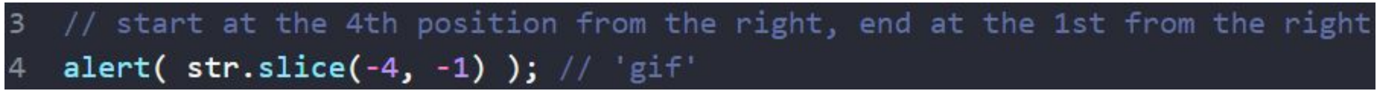
```
let str = "stringify";
alert( str.slice(0, 5) ); // 'strin', the substring from 0 to 5 (not including 5)
alert( str.slice(0, 1) ); // 's', from 0 to 1, but not including 1, so only character at 0
```

If there is no second argument, then slice goes till the end of the string:



```
let str = "stringify";
alert( str.slice(2) ); // 'ringify', from the 2nd position till the end
```

Negative values for start/end are also possible. They mean the position is counted from the string end:

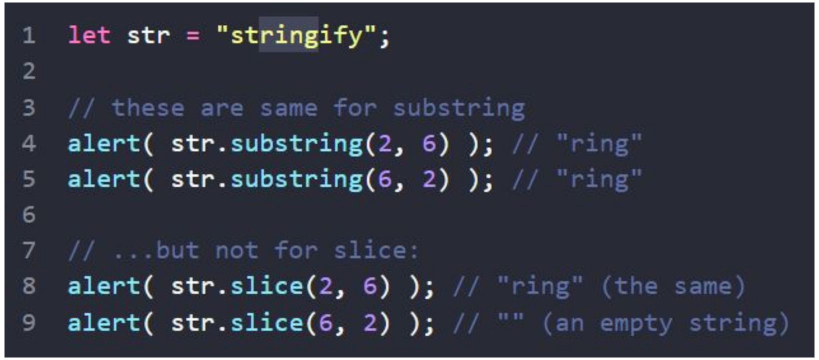


```
// start at the 4th position from the right, end at the 1st from the right
alert( str.slice(-4, -1) ); // 'gif'
```

String - Getting a substring (substring)

str.substring(start [, end])

Returns the part of the string between start and end (not including end).



```
let str = "stringify";

// these are same for substring
alert( str.substring(2, 6) ); // "ring"
alert( str.substring(6, 2) ); // "ring"

// ...but not for slice:
alert( str.slice(2, 6) ); // "ring" (the same)
alert( str.slice(6, 2) ); // "" (an empty string)
```

Negative arguments are (unlike slice) not supported, they are treated as 0.

String - Getting a substring (substr)

str.substr(start [, length])

Returns the part of the string from start, with the given length

In contrast with the previous methods, this one allows us to specify the length instead of the ending position:

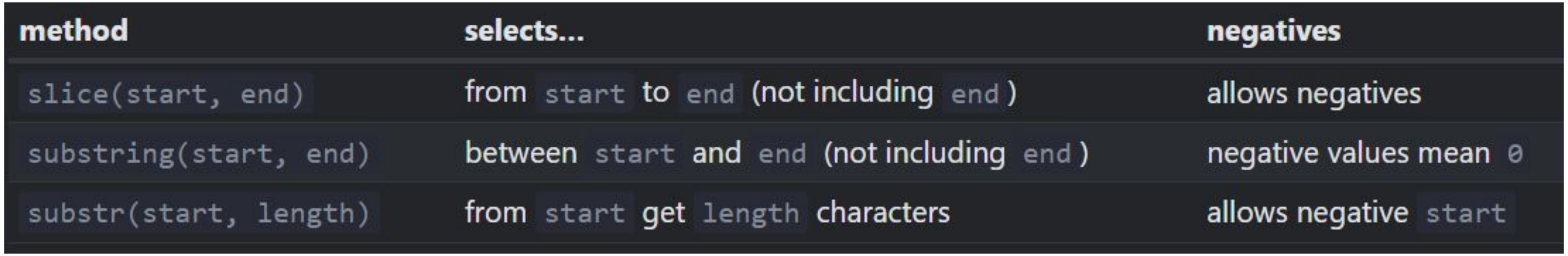
```
let str = "stringify";
alert( str.substr(2, 4) ); // 'ring', from the 2nd position get 4 characters
```

The first argument may be negative, to count from the end:

```
let str = "stringify";
alert( str.substr(-4, 2) ); // 'gi', from the 4th position get 2 characters
```

String - Getting a substring

Only remembering slice is enough for practical use. All these methods do the same thing essentially.



method	selects...	negatives
`slice(start, end)`	from `start` to `end` (not including `end`)	allows negatives
`substring(start, end)`	between `start` and `end` (not including `end`)	negative values mean `0`
`substr(start, length)`	from `start` get `length` characters	allows negative `start`

String - Comparing Substring

Strings are compared character-by-character in alphabetical order.

A lowercase letter is always greater than the uppercase: