What is String.slice()?

The slice() method extracts a section of a string and returns it as a new string, without modifying the original string. It is widely used for substring extraction due to its flexibility with indices.

Syntax of slice()

string.slice(startIndex [, endIndex])

Parameters of slice()

startIndex (required): The index at which to start extracting the substring.

If positive: Counts from the beginning of the string (e.g., 0 = first character).

If negative: Counts from the end of the string (e.g., -1 = last character).

endIndex (optional): The index at which to stop extracting (exclusive). If omitted, slice() extracts to the end of the string.

If positive: Stops before this index.

If negative: Stops before the index counted from the end.

Return Value of slice()

A new string containing the extracted substring. If startIndex is greater than or equal to endIndex, an empty string ("") is returned.

Examples of slice()#

Let’s see slice() in action with different scenarios:

Example 1: Basic extraction with positive indices#

```
const str = "Hello, World!";
```

```
console.log(str.slice(7, 12)); // "World" (starts at 7, ends before 12)
```

Example 2: Omitted endIndex (extracts to the end)

```
console.log(str.slice(7)); // "World!" (starts at 7, goes to the end)
```

Example 3: Negative startIndex (counts from the end)

```
console.log(str.slice(-6)); // "World!" (starts at index 7, since length is 13; 13 - 6 = 7)
```

Example 4: Negative endIndex (stops before the end)

```
console.log(str.slice(0, -1)); // "Hello, World" (stops before the last character `!`)
```

Example 5: startIndex ≥ endIndex (returns empty string)

console.log(str.slice(5, 3)); // "" (start > end, so no characters extracted)

What is String.substring()

The substring() method is another way to extract a substring from a string. Like slice(), it returns a new string and does not modify the original. However, its behavior differs in key ways, especially with negative indices and swapped start/end values.

Syntax of substring()

string.substring(indexStart [, indexEnd])

Parameters of substring()

indexStart (required): The index at which to start extracting the substring (inclusive).

If negative: Treated as 0 (unlike slice()).

indexEnd (optional): The index at which to stop extracting (exclusive). If omitted, substring() extracts to the end of the string.

If negative: Treated as 0.

If indexStart > indexEnd: The parameters are swapped (e.g., substring(5, 2) becomes substring(2, 5)).

Return Value of substring()#

A new string containing the extracted substring. If indexStart and indexEnd are the same, an empty string is returned.

Examples of substring()

Let's test substring() with similar scenarios to slice():

Example 1: Basic extraction with positive indices

const str = "Hello, World!";

console.log(str.substring(7, 12)); // "World" (same as slice() here)

Example 2: Omitted endIndex (extracts to the end)

console.log(str.substring(7)); // "World!" (same as slice() here)

Example 3: Negative indexStart (treated as 0)

console.log(str.substring(-6)); // "Hello, World!" (starts at 0, extracts entire string)

Example 4: Negative indexEnd (treated as 0)

console.log(str.substring(0, -1)); // "" (ends at 0, so no characters extracted)

Example 5: indexStart > indexEnd (parameters are swapped)

```
console.log(str.substring(5, 3)); // "llo" (swapped to substring(3,5), extracts from 3 to 5)
```

Key Differences Between slice() and substring()

Now that we’ve covered the basics, let’s break down the critical differences between slice() and substring().

1. Handling Negative Indices

This is the most significant difference:

slice(): Treats negative indices as offsets from the end of the string. For example, slice(-3) starts at the 3rd character from the end.

substring(): Ignores negative indices and treats them as 0. For example, substring(-3) is equivalent to substring(0).

Example:

```
const str = "JavaScript";

console.log(str.slice(-4)); // "ript" (starts at index 6: 10 - 4 = 6)

console.log(str.substring(-4)); // "JavaScript" (treated as substring(0))
```

2. Behavior When start > end

slice(): Returns an empty string ("") if startIndex is greater than or equal to endIndex.

substring(): Swaps indexStart and indexEnd if indexStart > indexEnd, effectively treating the smaller value as the start and the larger as the end.

Example:

```
const str = "abcdef";

console.log(str.slice(4, 2)); // "" (start=4 > end=2 → empty string)

console.log(str.substring(4, 2)); // "cd" (swapped to substring(2,4))
```

3. Treatment of Non-Integer and NaN Parameters

Both methods coerce non-integer parameters to integers, but there are nuances:

Non-integers: Both slice() and substring() floor (truncate towards zero) non-integer indices. For example, slice(2.9) is treated as slice(2).

NaN: Both treat NaN as 0. For example, slice(NaN) or substring(NaN) is equivalent to slice(0) or substring(0).

Example:

```
const str = "Hello";

console.log(str.slice(1.8, 3.2)); // "el" (coerced to slice(1,3))

console.log(str.substring(NaN, 3)); // "Hel" (coerced to substring(0,3))
```

4. Edge Cases

Omitted endIndex: Both methods extract from start to the end of the string.

start or end beyond string length: Both clamp the index to the string length. For example, slice(100) on a 5-character string returns "".

start = end: Both return an empty string.

When to Use slice() vs substring()

Use slice() when:

You need to extract substrings from the end of the string (negative indices), or you want predictable behavior when start > end (returns empty string). It's more flexible for most real-world scenarios.

Use substring() when:

You accidentally swap start and end indices and want the method to "fix" it (by swapping them), or you're working with legacy code that relies on its behavior. However, this is rare—slice() is generally preferred.

substr()

string.substr(start,length) The index of the first character is 0, start is required and length is optional

string.substring(start,length) The index of the first character is 0, start is required, end is optional

Example

Same Output: //for substr() and substring()

```
var str = "hello Tony";

str.substr(6);  //Tony

str.substring(6); //Tony
```

Difference:

When there are two parameters

(1) substr(start,length) returns from the start positionlengthSubstring

 “goodboy”.substr(1,6); //oodboy

[Note] When length is 0 or negative, an empty string is returned.

```
  "goodboy".substr(1,0);   //""
```

If it is a negative number, then the parameter declares the position from the end of the string. In other words, -1 refers to the last character in the string, -2 refers to the penultimate character, and so on.

```
"goodboy".substr(-1)        // "y"
```

(2) substring(start,end) returns the substring from the start position to the end position (not including end)

```
“goodboy”.substring(1,6);  //oodbo
```