String: Cadena de Caracteres en JavaScript

En el mundo de la programación, las cadenas de caracteres (o strings) son como los bloques de construcción del lenguaje. Representan datos textuales y son fundamentales para muchas aplicaciones. En este artículo, exploraremos los conceptos clave relacionados con las cadenas en JavaScript.

Creación de Strings

Una cadena es una colección de caracteres, como letras y símbolos. En JavaScript, podemos crear un literal de cadena encerrando un grupo de caracteres entre comillas simples (`' '`) o dobles (`" "`) o comillas inversas (`` ` ` ``) . Estas se denominan notación literal.

```
const user1 = 'Juan Pérez';
const user2 = "María López";
const user3 = `Carlos Rodríguez`;
```

ES6 introdujo literales de plantilla que permiten definir una cadena de caracteres de acento grave (`` ` ``): Esto potencializa el uso de `strings` en JavaScript, perolo vemos a profundidad en otro tutorial.

Además, las cadenas pueden contener una amplia gama de caracteres, incluyendo letras, números, signos de puntuación y símbolos especiales. Son comúnmente utilizadas para almacenar y manipular datos textuales en aplicaciones JavaScript.

Constructor String

Además de crear cadenas utilizando la notación literal, JavaScript proporciona un constructor de objeto llamado `String()` para crear objetos de tipo cadena. Por ejemplo:

```
let myString = new String('Hola mundo');
console.log(myString); // "Hola mundo"
```

En este ejemplo, estamos utilizando el constructor `String()` para crear un nuevo objeto de cadena con el valor `"Hola mundo"`. Es importante tener en cuenta que aunque esto crea un objeto de cadena, en la práctica, es más común y preferible usar la notación literal de cadena, debido a su simplicidad y claridad.

Escapando Caracteres Especiales

A veces, cuando se necesita incluir caracteres especiales dentro de una cadena, se puede utilizar la barra invertida (`\`) para escaparlos. Esto

permite asegurarse de que JavaScript interprete correctamente los caracteres especiales y los trate como parte del contenido de la cadena.

```
const mensaje = 'Ejemplo comilla simple (\') escapada';

console.log(mensaje); // Ejemplo comilla simple (') escapada
```

Otro ejemplo sería:

```
const str = 'I\'m a string!';
// "I'm a string"!
```

Además de las comillas simples y dobles, la barra invertida se utiliza para escapar otros caracteres especiales, como barras invertidas, retornos de carro, saltos de línea y caracteres de tabulación. Esto es útil cuando necesitas incluir estos caracteres en tus cadenas de texto de una manera que JavaScript pueda entender correctamente.

- Una barra invertida: `\\`
- Un retorno de carro: `\r`
- Un salto de línea: `\n`
- Un caracter de tabulación: `\t`

Obtener la longitud de un string

Para obtener la longitud de un `string` o una cadena en JavaScript, puedes usar la propiedad `length`.:

```
const message = 'Hola mundo!';
const txtlength = message.length;

console.log(txtlength); // 11
```

En este ejemplo, la variable `message` contiene la cadena `'Hola mundo!'`. Luego, utilizamos la propiedad `length` para obtener la longitud de la cadena, que en este caso es 11, y lo almacenamos en la variable `txtlength`. Finalmente, imprimimos la longitud en la consola.

Es importante tener en cuenta que en JavaScript existe el tipo String (con la primera letra en mayúscula), el cual es el tipo de contenedor primitivo para las cadenas. Esto significa que se pueden acceder a todas las propiedades y métodos del tipo String desde una cadena primitiva.

Acceder a los Caracteres de un string

En JavaScript, las cadenas de caracteres son similares a los arreglos en cuanto a que puedes acceder a cada carácter individual utilizando la notación de corchetes `[]`:

- Cada carácter en una cadena tiene un índice, comenzando por `0` para el primer carácter.

- Puedes usar este índice con corchetes para obtener el carácter en esa posición específica.
- Es importante recordar que las cadenas en JavaScript son **inmutables**, lo que significa que no puedes cambiar un carácter directamente usando esta notación.

```
const message = "Hola Mundo!";
console.log(message[0]); // "H"
console.log(message[5]); // "M"
```

Para acceder al último carácter de una cadena, puedes utilizar el índice `length - 1`.

```
const message = "Hola Mundo!";
console.log(message[message.length -1]); // "!"
```

Concatenación de strings usando el Operador `+`

En JavaScript, puedes unir (concatenar) dos o más cadenas utilizando el operador `+`. Esto te permite combinar fragmentos de texto para formar una cadena más larga. Aquí tienes algunos ejemplos:

Concatenación Básica:

```
const firstName = "Juan";
const lastName = "Pérez";
const fullName = firstName + " " + lastName;
console.log(fullName); // "Juan Pérez"
```

Concatenación con Variables y Texto Fijo:

```
const product = "Camisas";
const quantity = 3;
const message = "Tienes " + quantity + " " + product + " en tu carrito";
console.log(message); // "Tienes 3 Camisas en tu carrito"
```

Si deseas ensamblar una cadena pieza por pieza, puedes utilizar el operador `+=`. Por ejemplo:

```
let className = 'btn';
className += ' btn-primary';
className += ' none';

console.log(className); // "btn btn-primary none"
```

Convertir Valores a String

Convertir valores a cadena es útil cuando necesitas representar datos en forma de texto. En JavaScript, puedes hacerlo utilizando el método `toString()` o la función `String()`.

Usando el método `toString()`:

```
let num = 123;
let strNum = num.toString();
console.log(strNum); // "123"
```

Usando la función `String()`

```
let booleanValue = true;
let strBool = String(booleanValue);
console.log(strBool); // "true"
```

Ambos ejemplos muestran cómo convertir un número y un booleano a cadena utilizando `toString()` y `String()` respectivamente.

Es importante destacar que el método `toString()` no funciona para valores `undefined` y `null`. Ademas, cuando se convierte una cadena a booleano, no se puede revertir el proceso.

Comparando strings o cadenas

Para comparar dos cadenas se utilizan operadores de comparación como `>`, `>=`, `<`, `<=` y `==` o el método `localeCompare()`

Los operadores de comparación comparan cadenas según los valores numéricos de los caracteres. Y puede devolver un orden de cadena diferente al utilizado en los diccionarios. Por ejemplo:

```
let result = 'a' < 'b';

console.log(result); // true
```

Veamos el siguiente ejemplo:

```
const str1 = 'apple';
const str2 = 'banana';

console.log(str1 < str2); // true
```

El anterior código devuelve true, porque ‘apple’ viene antes de ‘banana’ alfabéticamente.

También puedes usar el método `localeCompare()`, que devuelve un número negativo si la primera cadena viene antes en orden alfabético, un número positivo si viene después y 0 si son iguales. Por ejemplo:

```
const str1 = 'apple';
const str2 = 'banana';

console.log(str1.localeCompare(str2)); // -1
```

El anterior ejemplo devuelve `-1`, porque ‘apple’ viene antes de ‘banana’ alfabéticamente.

Otros Métodos Útiles

Las cadenas de caracteres, o strings, en JavaScript ofrecen una variedad de métodos integrados para su manipulación. Entre ellos se

encuentran `split()`, `substring()`, `indexOf()`, `toUpperCase()`, `toLowerCase()`, entre otros.