HINTS

COMPARACIÓN ENTRE REPLACE() Y REPLACEALL()

En el transcurso de este CUE, aprendimos algunas funciones que pueden ser usadas con cualquier objeto de tipo String. Uno de estos métodos fue: `replaceAll()`.

Antes de ES.NEXT, el prototipo del objeto String solo tenía el método `replace()`. Éste es muy similar al que aprendimos, y en estas sugerencias aclararemos, en parte, las similitudes y diferencias entre el método `replaceAll()`, y otras variaciones similares como `replace()`.

Los métodos `replaceAll()` y `replace()` funcionan de la misma manera, salvo 2 cosas:

I. Si el argumento de búsqueda es una cadena, `replaceAll()` reemplaza todas las apariciones de la búsqueda; mientras que `replace()` solamente la primera ocurrencia.
II. Si el argumento de búsqueda es una expresión regular no global, `replaceAll()` lanza una excepción `TypeError`.

Aunque la diferencia es bastante sutil, conocerlas puede ser útil para depurar si nuestros errores se deben a que nuestras funciones no están resultando, o si no las estamos usando correctamente.

CORTOCIRCUITO

Si pensamos en un cortocircuito en términos eléctricos, parte de una definición describe este evento como: "una desviación de la corriente". En sentido figurado, sucede algo semejante en JavaScript cuando utilizamos los operadores de asignación lógica, y por esta razón al suceso se le denomina un ”cortocircuito”.

En el cortocircuito de JavaScript, una expresión se evalúa de izquierda a derecha, hasta que se confirma que el resultado de las condiciones restantes no afectará al que ya está evaluado.

Antes de ES.NEXT, se consideraba que solo los operadores lógicos AND `&&` y OR `||` experimentaban el comportamiento de cortocircuito, pero también se pueden incluir los operadores lógicos de asignación, a la lista de aquellos que lo realizan.

Las evaluaciones que siguen el comportamiento de cortocircuitos evitan trabajo innecesario, y conducen a procesamientos más eficientes en nuestras aplicaciones.

EL OBJETO STRING

Permite trabajar con una serie de caracteres; envolviendo el tipo de datos primitivo de cadena de JavaScript con una serie de métodos auxiliares. Las cadenas son útiles para almacenar datos que se pueden representar en forma de texto.

Algunas de las operaciones más utilizadas en cadenas son: verificar su longitud, construirlas y concatenarlas, usando los operadores "`+`" y "`+=`".

Las cadenas se pueden crear como primitivas, a partir de cadenas literales o como objetos, utilizando el constructor `String()`:

```
const cadena1 = 'una cadena primitiva';
const cadena2 = `otra cadena primitiva`;

const cadena3 = new String("Un objeto cadena o String");
```

Las primitivas de cadena, y los objetos de cadena, se pueden emplear indistintamente en la mayoría de las situaciones.

Los literales de cadena pueden especificarse utilizando comillas simples (' ') o dobles (" "), que se tratan de manera idéntica, o empleando el carácter de acento grave (`). Esta última especifica *un literal de plantilla*: aquel que puede interpolar expresiones usando texto plano, y variables entre corchetes y un signo dólar.

```
let nombre = "Aquiles"
const cadena4 = `Mi nombre es ${nombre}`;
```