Somos un ecosistema de desarrolladores de software

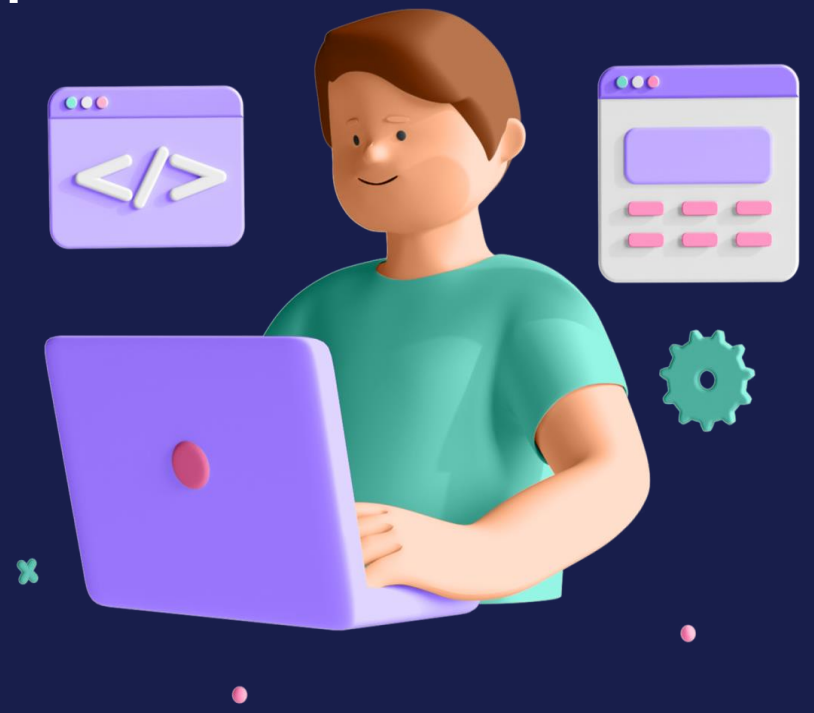
</Riwi>

Introducción a Java Script

String

Propiedades

- Comillas simples: '
- Comillas dobles: "
- Backticks: ` (ver más adelante, en Interpolación de variables)



String

Propiedades

```
// Notación literal (preferida)
const text = "¡Hola a todos!";
const message = "Otro mensaje de texto";

// Notación mediante objeto
const texts = new String("¡Hola a todos!");
const messages = new String("Otro mensaje de texto");
```

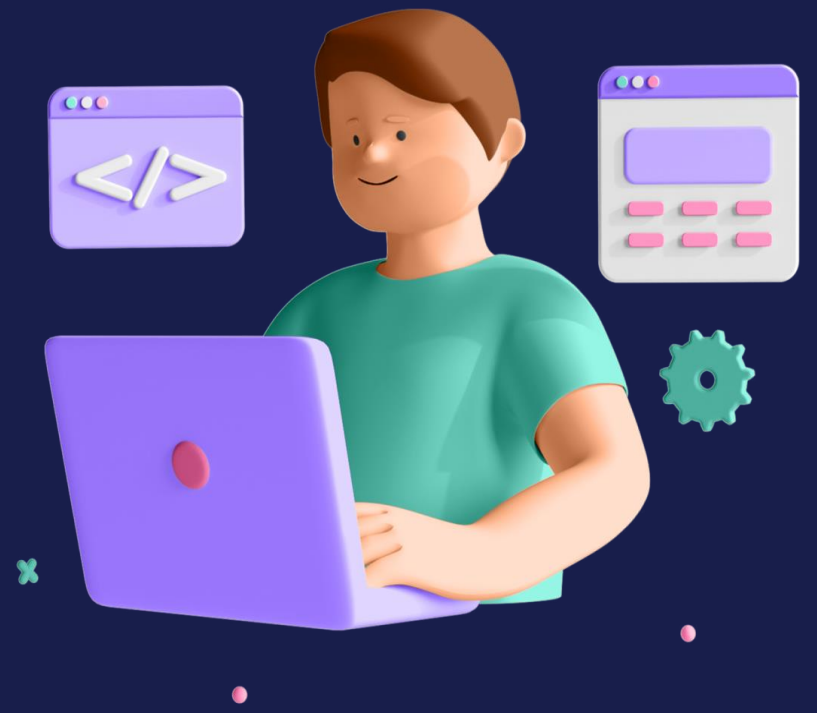
Constructor	Descripción
STRING new String(text)	Crea un objeto de texto a partir del texto text pasado por parámetro. Evitar
STRING "texto"	Simplemente, escribimos el texto entre comillas simples o dobles. Notación preferida.

String

Propiedades

Propiedad	Descripción
.length	Devuelve el número de carácteres totales del texto.

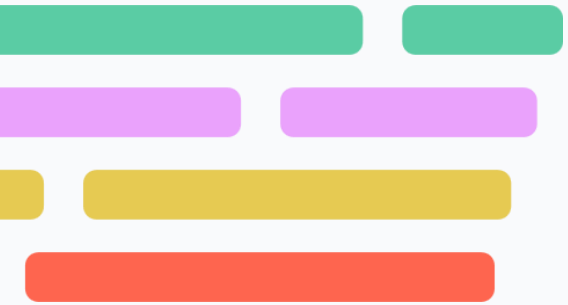
```
console.log("Hola".length);     // 4
console.log("Adiós".length);    // 5
console.log("".length);         // 0
console.log("¡Yeah!".length);   // 6
```



String

Acceso a un carácter

El texto "Hola", esta formado por los caracteres "H", "o", "l" y "a". Si queremos acceder a cada uno de ellos, podemos utilizar el operador [] indicando la posición a la que queremos acceder:



```
const text = "Hola";

text[0];      // "H"
text[1];      // "o"
text[2];      // "l"
text[4];      // undefined
```

String

Interpolación de variables

```
const word1 = "mejores ";
const word2 = "son ";
const word3 = "Riwi.";

console.log("Los " + word1 + "Coders " + word2 + "de "+ word3 );
```

```
const word1 = "mejores";
const word2 = "son";
const word3 = "Riwi.";

console.log(`Los ${word2} Coders ${word2} de ${word3}`);
```

String

Interpolación de variables

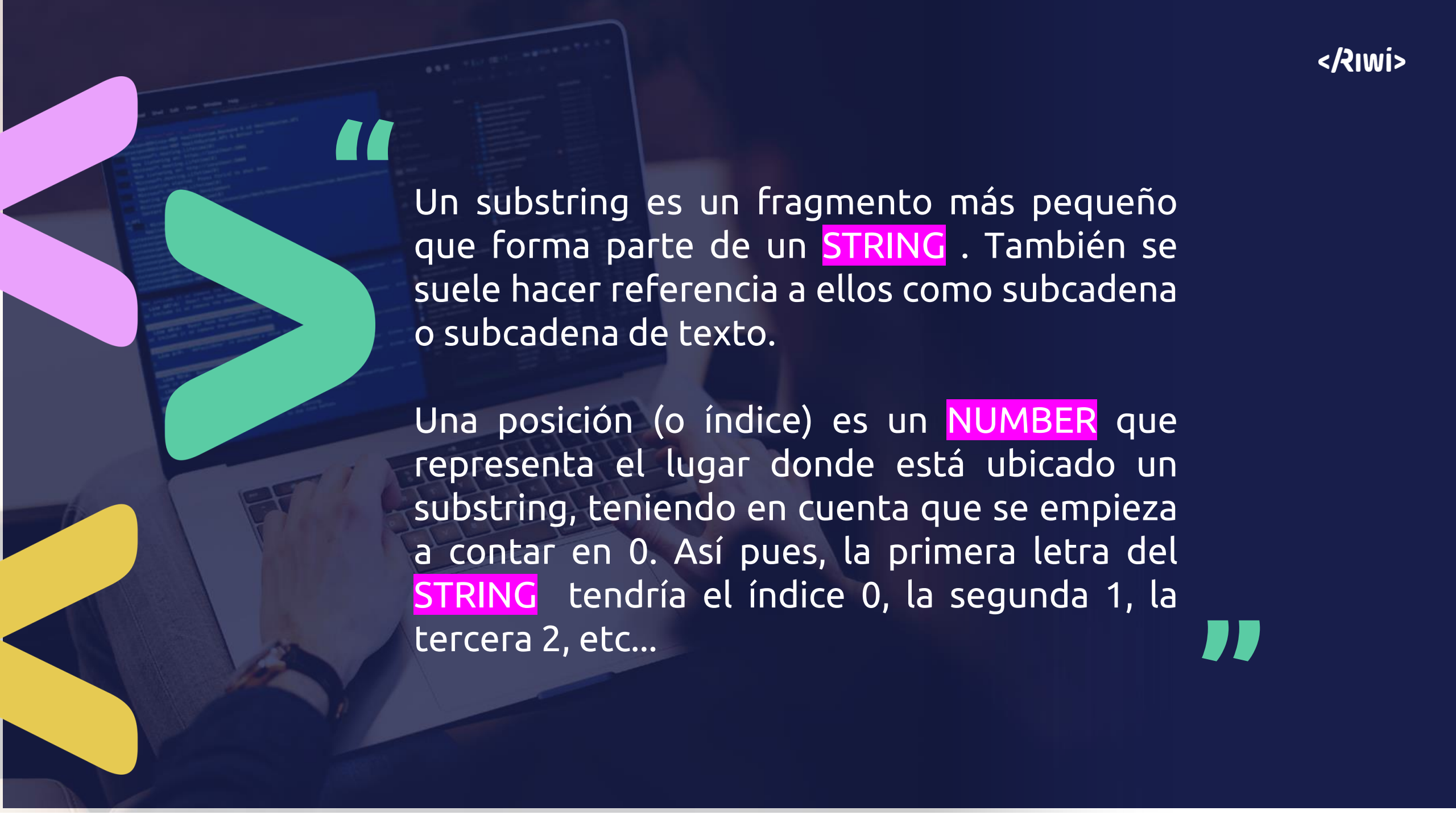
- Permite **múltiples líneas**, algo que no se puede hacer con las demás comillas
- Permite interpolar expresiones Javascript (no sólo variables)
- Permite interpolar el valor de variables (ya mencionado)

String

Interpolación de variables

Observa que template incluye el código HTML de magicalWord, algo que quizás no tiene mucho sentido aún, pero que puede cobrar mucho sentido si pensamos en crear funciones reutilizables

```
const magicalWord = `<strong>Magical Word</strong>`;
const template = `
  <div class="container">
    ${magicalWord}
  </div>
`;
```



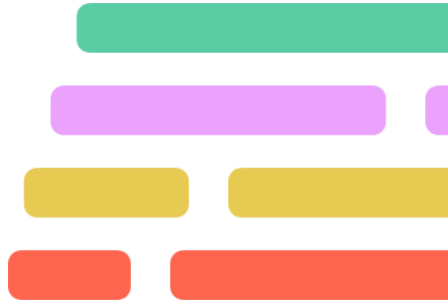
Un substring es un fragmento más pequeño que forma parte de un STRING . También se suele hacer referencia a ellos como subcadena o subcadena de texto.

Una posición (o índice) es un NUMBER que representa el lugar donde está ubicado un substring, teniendo en cuenta que se empieza a contar en 0. Así pues, la primera letra del STRING tendría el índice 0, la segunda 1, la tercera 2, etc...

Posiciones y substrings

Obtener posición o índice

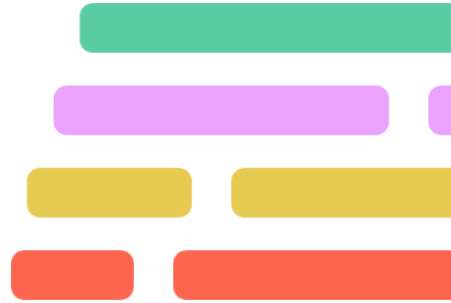
Método	Descripción
.charAt(pos)	Devuelve el carácter de la posición pos. Similar al operador [].
.indexOf(text)	Devuelve la primera posición del texto text.
.indexOf(text, from)	Idem al anterior, partiendo desde la posición from.
.lastIndexOf(text)	Devuelve la última posición del texto text.
.lastIndexOf(text, from)	Idem al anterior, partiendo desde from hacia el inicio.



Posiciones y substrings

Carácter en cierta posición

```
let myName = "Manz";

// Utilizando .charAt
console.log(myName.charAt());    // 'M'
console.log(myName.charAt(0));   // 'M'
console.log(myName.charAt(1));   // 'a'
console.log(myName.charAt(10));  // ''

// Utilizando operador []
console.log(myName[0]);          // 'M'
console.log(myName[1]);          // 'a'
console.log(myName[10]);         // undefined
```



Posiciones y substrings

Posición de cierto carácter

```
const phrase = "LenguajeJS, página de Javascript";

console.log(phrase.indexOf("n"));          // 2
console.log(phrase.indexOf("n", 3));       // 16
console.log(phrase.indexOf("n", 17));      // -1
```

Posiciones y substrings

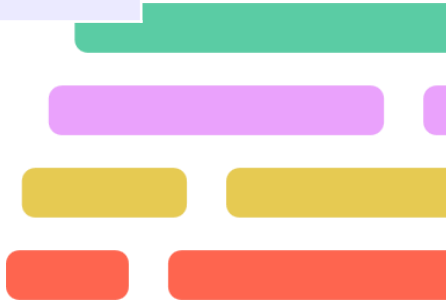
Posición desde el final

```
const phrase = "LenguajeJS, página de Javascript";

console.log(phrase.lastIndexOf("a"));      // 14
console.log(phrase.lastIndexOf("a", 8));    // 5
console.log(phrase.lastIndexOf("a", 5))   //5
```

Posiciones y substrings

Obtener fragmentos (substrings)

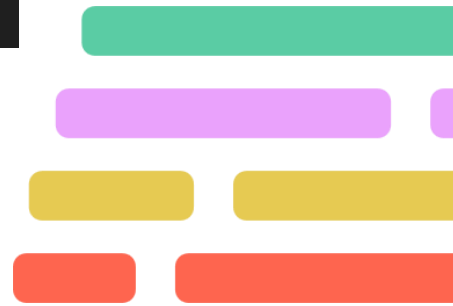
Método	Descripción
.repeat(num)	Devuelve el repetido num veces.
.substring(start, end)	Devuelve el substring desde la posición start hasta end.
.substr(start, size)	Devuelve el substring desde la posición start hasta start+size.
.slice(start, end)	Idem a .substr() con leves diferencias.

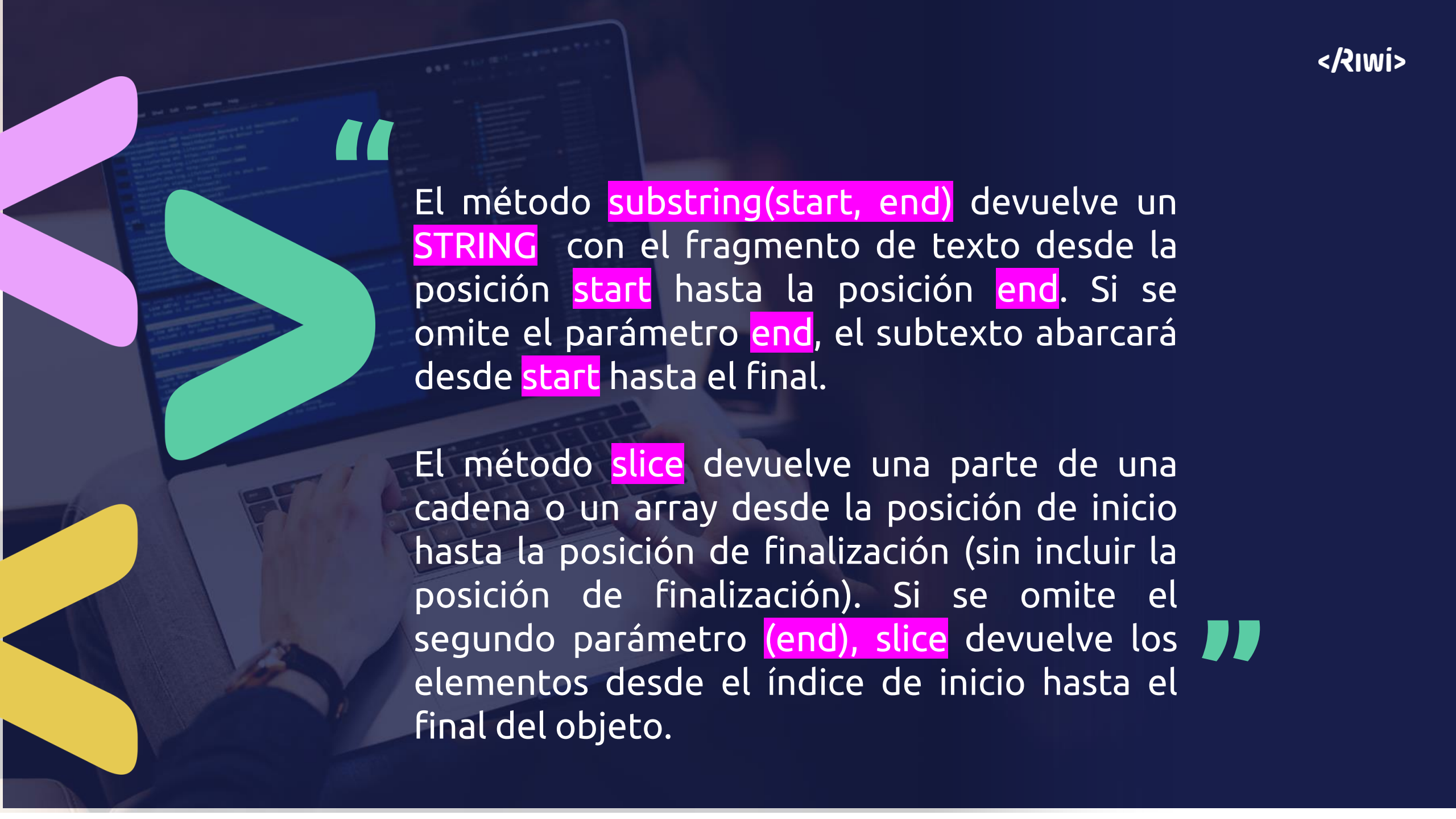


Posiciones y substrings

Repetir cadena de texto

```
const text = "Riwi ";

console.log(text.repeat(2));  // Riwi Riwi
console.log(text.repeat(3));  // Riwi Riwi Riwi
console.log(text.repeat(4));  // Riwi Riwi Riwi
console.log(text.repeat());   //
console.log(text.repeat(-1)); // ERROR
```





El método substring(start, end) devuelve un STRING con el fragmento de texto desde la posición start hasta la posición end. Si se omite el parámetro end, el subtexto abarcará desde start hasta el final.

El método slice devuelve una parte de una cadena o un array desde la posición de inicio hasta la posición de finalización (sin incluir la posición de finalización). Si se omite el segundo parámetro (end), slice devuelve los elementos desde el índice de inicio hasta el final del objeto.

Posiciones y substrings

</Riwi>

Fragmento de texto (substring)

```
const text = "koenigsegg";

// Utilizando substring
console.log(text.substring(3));      // 'nigsegg' (desde el 3 en adelante)
console.log(text.substring(3, 5));   // 'ni'      (desde el 3, hasta el 5)

// Utilizando slice
console.log(text.slice(3));          // 'nigsegg' (desde el 3 en adelante)
console.log(text.slice(3, 8));       // 'nigse'   (desde el 3, hasta el 8)
console.log(text.slice(-3));         // 'egg'      (desde la posición 3 desde el final, en adelante)
console.log(text.slice(-3, -1));     // 'eg'       (desde la posición 3 desde el final, hasta 1 posición antes del final)
```

Posiciones y substrings

Dividir un texto en partes (array)

Método	Descripción
.split(text)	Separa el texto en varias partes, usando text como separador.
.split(text, limit)	Idem, pero crea como máximo limit fragmentos.
.split(regexp)	Separa el texto usando la regexp como separador.
.split(regexp, limit)	Idem, pero crea como máximo limit fragmentos.

Posiciones y substrings

Dividir un texto en partes (array)

STRING como separador

```
"88.12.44.123".split(".");     // ["88", "12", "44", "123"] (4 elementos)
"1.2.3.4.5".split(".");        // ["1", "2", "3", "4", "5"] (5 elementos)
"Hola a todos".split(" ");     // ["Hola", "a", "todos"] (3 elementos)
"A,B,C,D,E".split(",", 3);     // ["A", "B", "C"] (limitado a los 3 primeros elementos)
"Código".split("");            // ["C", "ó", "d", "i", "g", "o"] (6 elementos)
```

Posiciones y substrings

Dividir un texto en partes (array)

STRING como separador

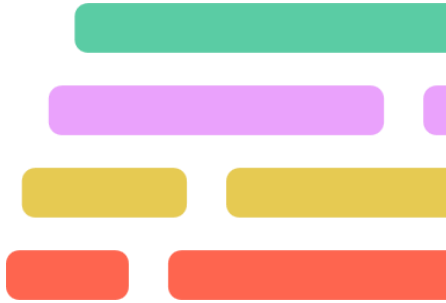
```
"88.12.44.123".split(".");     // ["88", "12", "44", "123"] (4 elementos)
"1.2.3.4.5".split(".");        // ["1", "2", "3", "4", "5"] (5 elementos)
"Hola a todos".split(" ");     // ["Hola", "a", "todos"] (3 elementos)
"A,B,C,D,E".split(",", 3);     // ["A", "B", "C"] (limitado a los 3 primeros elementos)
"Código".split("");            // ["C", "ó", "d", "i", "g", "o"] (6 elementos)
```

```
// Separa tanto por punto como por coma
"88.12,44.123".split(/[.,]/);    // ["88", "12", "44", "123"] (4 elementos)
```

Buscar y reemplazar

Busqueda y/o reemplazo

- **Comprobación** : La más ligera de las tres. Sólo comprueba si existe el fragmento de texto.
- **Búsqueda** : Busca un fragmento de texto, devolviendo información del texto encontrado (posición, texto...).
- **Reemplazo** : Realiza una búsqueda de un texto y además un reemplazo. Suele ser más costoso.

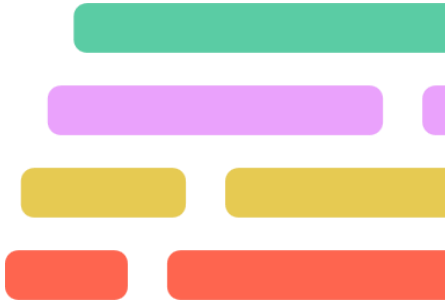


Buscar y reemplazar

Comprobación en textos

Método	Descripción
BOOLEAN .startsWith(text, from)	Comprueba si el texto comienza por text.
BOOLEAN .endsWith(text, to)	Comprueba si el texto termina por text.
BOOLEAN .includes(text, from)	Comprueba si el texto contiene el subtexto text.

El método .startsWith() devolverá true si el comienza por text. De lo contrario, false.
El método .endsWith() devolverá true si el acaba en text. De lo contrario, false.
El método .includes() devolverá true si el contiene text. De lo contrario, false.



Buscar y reemplazar

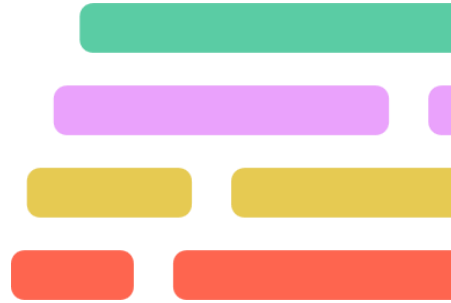
Búsqueda de cadenas de textos

```
const text = "Manz";

text.startsWith("M");      // true  ('Manz' empieza por 'M')
text.startsWith("a", 1);   // true  ('anz' empieza por 'a')
text.endsWith("o");        // false ('Manz' no acaba en 'o')
text.endsWith("n", 3);     // true  ('Man' acaba en 'n')
text.includes("an");       // true  ('Manz' incluye 'an')
text.includes("M", 1);     // false ('anz' no incluye 'M')
```

Buscar y reemplazar

Búsqueda de cadenas de textos

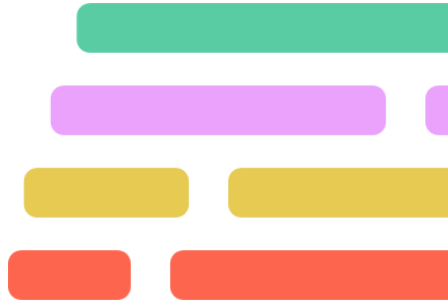
Método	Descripción
NUMBER .search(regexp)	Busca un patrón que encaje con regexp y devuelve la posición encontrada.
ARRAY .match(regexp)	Idem a la anterior, pero devuelve las coincidencias encontradas.
ARRAY .matchAll(regexp)	Idem a la anterior, pero devuelve un iterador para iterar por cada coincidencia.



Buscar y reemplazar

Búsqueda de cadenas de textos

El método .search() devuelve la posición de la primera ocurrencia. -1 si no se encuentra.
El método .match() devuelve un ARRAY con las coincidencias encontradas. null si no se encuentran.
El método .matchAll() devuelve un iterador para poder recorrer las coincidencias encontradas.



Buscar y reemplazar

Búsqueda de cadenas de textos

```
const text = "El gato, el perro y el pato.";
const regexp = /.a.o/g;

text.search(regexp);   // 3, porque la primera coincidencia ocurre en la posición 3 (gato)
text.match(regexp);    // ["gato", "pato"], las dos coincidencias encontradas
```

```
const text = "El gato, el perro y el pato.";
const regexp = /.a.o/g;

const iterator = text.matchAll(regexp);
for (let ocurrence of iterator) {
  console.log(ocurrence);
}

// ['gato', index: 3, input: 'El gato, el perro y el pato.', groups: undefined]
// ['pato', index: 23, input: 'El gato, el perro y el pato.', groups: undefined]
```

Buscar y reemplazar

Búsqueda de cadenas de textos

</Riwi>

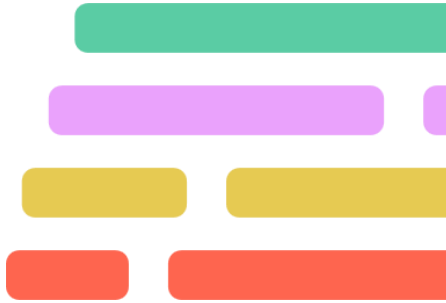
```
const text = "El gato, el perro y el pato.";
const regexp = /.a.o/g;
const results = [...text.matchAll(regexp)];    // ["gato", "pato"]

results.length      // 2
results[0].index    // 3
results[1].index    // 23
```

Buscar y reemplazar

Reemplazar cadenas de texto

Método	Descripción
STRING.replace(text, newText)	Reemplaza la primera aparición del STRING text por newText.
STRING .replace(regexp, newText)	Idem, pero busca a partir de una REGEXP en lugar de un STRING.
STRING .replaceAll(text, newText)	Reemplaza todas las apariciones del texto text por newText.
STRING .replaceAll(regexp, newText)	Idem, pero busca a partir de una REGEXP en lugar de un STRING.

El método replace() reemplaza solo la primera aparición de un texto (salvo que se use una regexp global)
El método replaceAll() reemplaza todas las apariciones de un texto.



Buscar y reemplazar

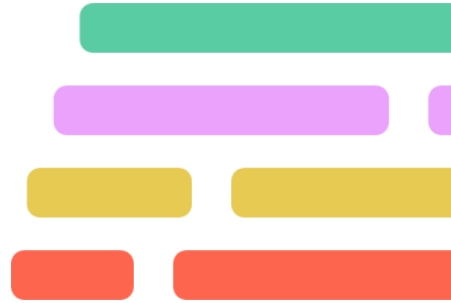
Reemplazar textos (.replace y .replaceAll())

```
const text = "Hola gato, ¿eres un perro o eres un pato?";

// Reemplazar la primera ocurrencia
text.replace("g", "p");                    // "Hola pato, ¿eres un perro o eres un pato?"
text.replace("g", "p").replace("o", "a");  // "Hala pato, ¿eres un perro o eres un pato?"

// Reemplazar todas las ocurrencias
text.replaceAll("e", "i");                 // "Hola gato, ¿iris un pirro o iris un pato?"
text.replace(/e/g, "i");                   // "Hola gato, ¿iris un pirro o iris un pato?"
```

Buscar y reemplazar

Reemplazar todos textos

```
const daenerys = "Devuelveme mi gansito";

console.log(daenerys.replace(/[aeou]/g, "i"));      // 'Diviilvimi mi ginsiti'
```



Buscar y reemplazar

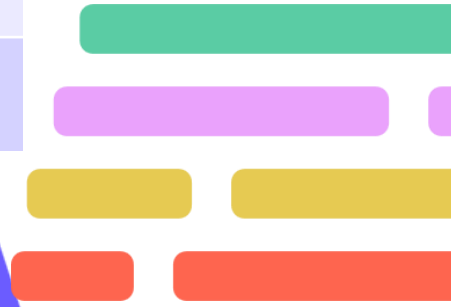
Función para reemplazar

```
const text = "Hola gato, ¿eres un perro o eres un pato?";
const replaceAction = (value) => `=>${value}<=`;

text.replace(/.a.o/g, replaceAction);
// "Hola =>gato<=, ¿eres un perro o eres un =>pato<=?"

text.replaceAll("un", replaceAction);
// "Hola gato, ¿eres =>un<= perro o eres =>un<= pato?"
```

Modificar cadenas de texto

Modificar Strings

Método	Descripción
.toLowerCase()	Devuelve el transformado a minúsculas.
.toUpperCase()	Devuelve el transformado a mayúsculas.
.padStart(size, text)	Devuelve el rellenando el inicio con text hasta llegar al tamaño size.
.padEnd(size, text)	Devuelve el rellenando el final con text hasta llegar al tamaño size.
.trimStart()	Devuelve el eliminando espacios a la izquierda del texto.
.trimEnd()	Devuelve el eliminando espacios a la derecha del texto.
.trim()	Devuelve el eliminando espacios a la izquierda y derecha del texto.



Modificar cadenas de texto

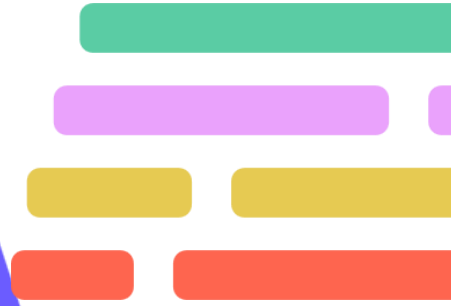
</Riwi>

Mayúsculas y minúsculas

```
const text = "Los gatos dominarán el mundo.";

text.toLowerCase();   // "los gatos dominarán el mundo."
text.toUpperCase();   // "LOS GATOS DOMINARÁN EL MUNDO."
```

Capitalizar texto

```
const text = "quiero capitalizar este texto.";

const toCapitalize = (text) => text[0].toUpperCase() + text.substring(1);
const toCapitalizeEveryWord = (text) => {
  return text.split(" ")                        // Separamos en un array cada palabra
    .map(word => toCapitalize(word))            // Aplicamos el capitalizar a cada elemento
    .join(" ");                                 // Lo volvemos a unir en un string
}

toCapitalize(text);             // "Los gatos dominarán el mundo."
toCapitalizeEveryWord(text);    // "Los Gatos Dominarán El Mundo."
```



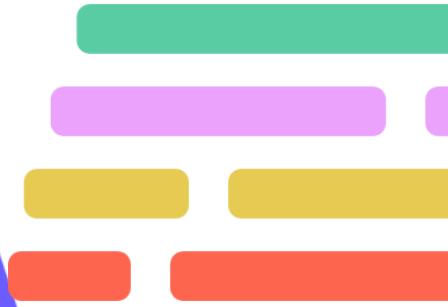
Modificar cadenas de texto

Relleno de cadenas

```
const stringNumber = "15";

stringNumber.padStart(5, "0");    // "00015"
stringNumber.padStart(3, "0");    // "015"
stringNumber.padStart(2, "0");    // "15"
stringNumber.padStart(1, "0");    // "15"
```

Eliminar espacios sobrantes

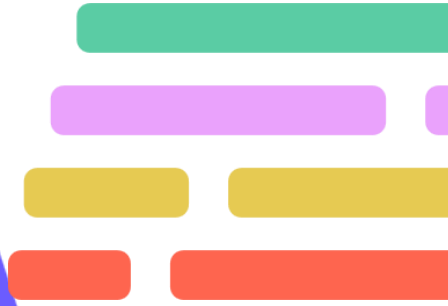
```
// Observa los espacios añadidos en los extremos de la frase
const text = "  ¡Saludad a vuestro nuevo rey gato!  ";

text.trim();         // "¡Saludad a vuestro nuevo rey gato!"
text.trimStart();    // "¡Saludad a vuestro nuevo rey gato!  "
text.trimEnd();      // "  ¡Saludad a vuestro nuevo rey gato!"

text.trimStart().trimEnd() === text.trim();     // true
```



Modificar cadenas de texto

Alternativas para crear (concatenar)

```
const myname = "Manz";

// Utilizando método concat
myname.concat("i", "to");      // "Manzito"
myname.concat(4, 5);           // "Manz45"
myname.concat((4 + 5));        // "Manz9"

// Utilizando operador +
myname + "i" + "to";           // "Manzito"
myname + 4 + 5;                // "Manz45"
myname + (4 + 5);              // "Manz9"
```



</Be a coder>