Alumno/a: Rafael Miguel Cruz Álvarez

Francisco Javier González Sabariego

Tarea 3.1. Presentación Objetos predefinidos en JavaScript: String, Regexp & Json

¿Qué es String?

Un String es un tipo de dato primitivo que es inmutable y que se utiliza para representar y manipular una secuencia de caracteres.

Forma de declarar

```
let cadena = new String("Creado con new String")
undefined
let cadena2 = "Creado sin constructor"
undefined
cadena
▶ String {"Creado con new String"}
cadena2
"Creado sin constructor"
typeof cadena
"object"
typeof cadena2
"string"
```

Hay dos formas de declarar un String, una como objeto y otra como dato primitivo.

Si se declara como un objeto, hay que usar la palabra reservada new String() y para declararlo como un dato primitivo debemos de usar las comillas dobles.

Propiedades

Length: La propiedad length de un objeto String representa la longitud de una cadena.

```
> let cadena = "Hola Mundo"
< undefined
> cadena.length
< 10
>
```

Propiedades

String[0]: Nos permite acceder indice querido de la cadena y así sacar el caracter de esa posición.

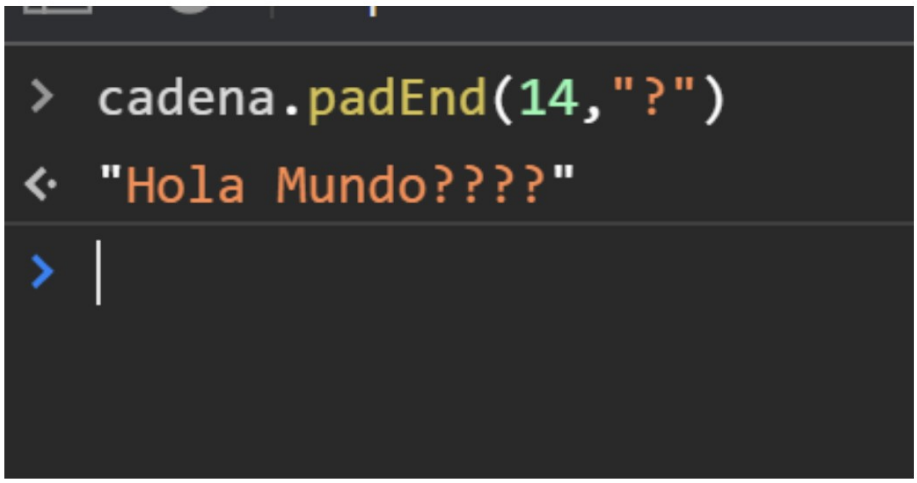
```
> let cadena = "Hola Mundo"
< undefined
> cadena[0]
< "H"
>
```

Métodos

padStart(): Método usado para rellenar la cadena actual con una cadena. Hay que indicar el tamaño total de la nueva cadena modificada, sino, no se modificará.

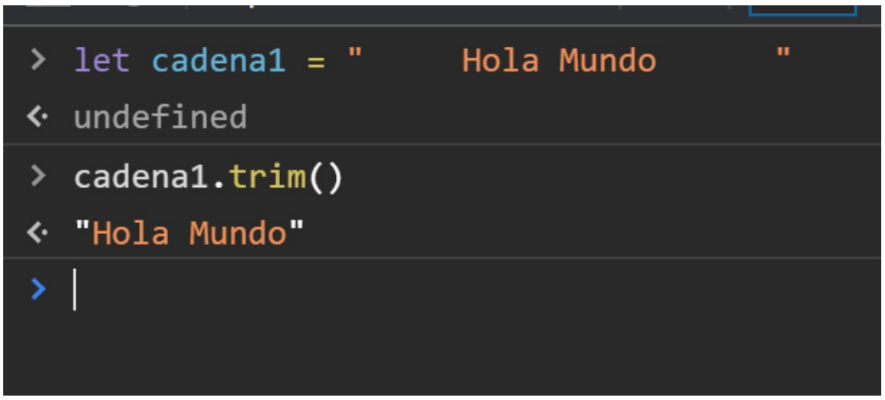
Métodos

padEnd(): Método similar que padStart() pero esta vez empezando a añadir desde el final.



Métodos

trim(): Método encargado de quitar los espacios en blanco tanto al inicio como al final de la cadena.



Métodos

trimEnd(): Método encargado de quitar los espacios en blanco del final de la cadena.

Métodos

trimStart(): Método encargado de quitar los espacios en blanco del inicio de la cadena.

Métodos

indexOf(): Método que devuelve el indice de la posición de la primera ocurrencia del valor especificado , si no se encuentra, devuelve -1. indexOf() es sensible a mayusculas, si no se encuentra sigue devolviendo -1.

Métodos

lastIndexOf(): Método que devuelve el índice de la última ocurrencia del valor especificado, si no se encuentra se devuelve -1. Es sensible a mayúsculas si no se encuentra devuelve -1.

```
> cadena
< "Hola Mundo"
> cadena.lastIndexOf("o")
< 9
> cadena.lastIndexOf("t")
< -1
> cadena.lastIndexOf("M")
< 5
> cadena.lastIndexOf("m")
< -1
>
```

Métodos

split(): Método que divide un objeto de tipo string en un array de cadenas separándola en subcadenas.

```
> cadena
< "Hola Mundo"
> cadena.split("l",1)
< ▶ ["Ho"]
> cadena.split("l",2)
< ▶ (2) ["Ho", "a Mundo"]
>
```

Métodos

slice(): Método con el que podemos extraer una parte de una cadena y devolverla en otra cadena nueva. Se puede indicar con números positivos para empezar a contar desde el inicio o desde el final con números negativos.

```
> cadena
< "Hola Mundo"
> cadena.slice(1,8)
< "ola Mun"
> cadena.slice(1,-2)
< "ola Mun"
>
```

Métodos

startWith(): Método en el que se indica si un String comienza con los caracteres de otro string, devolviendo true o false. Además de la cadena que se quiere comprobar podemos indicar la posición por la que deberia empezar, por defecto es la posición 0.

```
> cadena
< "Hola Mundo"
> cadena.startsWith("Hol")
< true
> cadena.startsWith("Mundo")
< false
> cadena.startsWith("ol",1)
< true
> cadena.startsWith("ol",2)
< false
>
```

Métodos

endWith(): Método que determina cuando una cadena termina con los caracteres de otra cadena, devolviendo true o false.

```
> cadena
< "Hola Mundo"
> cadena.endsWith("Mundo")
< true
> cadena.endsWith("Hola")
< false
> cadena.endsWith("Mundo",10)
< true
> cadena.endsWith("Mundo",9)
< false
> cadena.endsWith("mundo",10)
< false
>
```

Métodos

toUpperCase(): Método encargado de devolver la cadena en mayúsculas.

```
> cadena
< "Hola Mundo"
> cadena.toUpperCase()
< "HOLA MUNDO"
>
```

Métodos

toLowerCase(): Método encargado de devolver la cadena en minúsculas.

```
> cadena
"Hola Mundo"
> cadena.toLowerCase()
"hola mundo"
>
```

Métodos

concat(): Método usado para combinar dos o más cadenas.

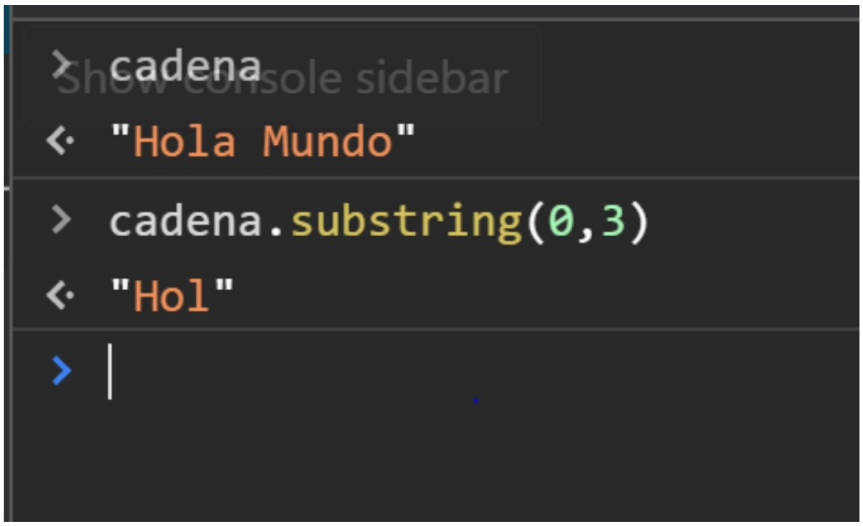
```
> cadena
"Hola Mundo"
> cadena3
" Como Estamos"
> cadena.concat(cadena3)
"Hola Mundo Como Estamos"
>
```

Métodos

include(): Método que determina si una cadena de texto se puede encontrar dentro de otra, devolviendo true o false. Sensible al uso de mayúsculas.

Métodos

substring(): Método que devuelve un subconjunto de un String.



Métodos

toString(): Método que devuelve el valor en cade de un objeto String.

```
> cadena2
◂ ▸ String {"Hola mundo"}
> cadena2.toString()
◂ "Hola mundo"
>
```

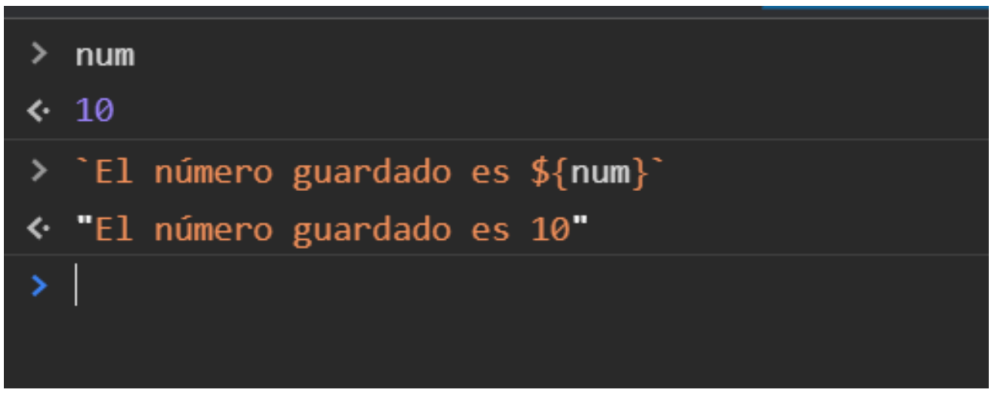
Métodos

replace(): Método que se usa para devolver una cadena nueva con alguna o todas las coincidencias de un valor siendo éstas coincidencias reemplazadas por otro valor.

```
> cadena
◂ "Hola Mundo"
> cadena.replace("Mundo","Buenas")
◂ "Hola Buenas"
>
```

Templates Literals

Las plantillas literales son cadenas literales que permiten el uso de expresiones incrustadas. Con ellas, se pueden usar cadenas de caracteres de más de una línea. Para crearlas es necesario usar las tildes invertidas(``). Para poder usar una variable dentro de estas es necesario usar lo siguiente: ${expresión/variable}.



¿Qué es una expresión regular?

Una expresión regular nos permite encontrar elementos o secciones coincidentes en una cadena respecto a un patrón definido.

Formas de crearlas

Existen dos formas de crear una RegExp:

- **Forma literal**
- **Usando el constructor new RegExp()**

```
let patternDni = /^(\d{8})[-\s]?(\w)$/i
undefined
let patternDni2 = new RegExp(/^(\d{8})[-\s]?(\w)$/, 'i')
undefined
let patternDni3 = new RegExp('^(\\d{8})[-\\s]?(\\w)$', 'i')
undefined
```

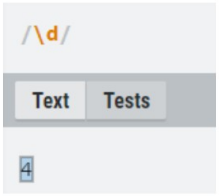
Caracteres especiales: Character classes.

/\w/ Word nos permite capturar cualquier letra, equivalente a [A-Za-z0-9_]:

/\W/ Not word al revés que el caso anterior, equivale a [^A-Za-z0-9_]:

Caracteres especiales: Character classes.

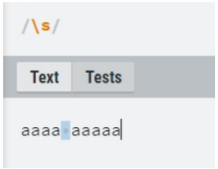
/\d/ Digit nos permite capturar cualquier dígito, equivalente a [0-9]:



/\D/ Not digit al revés que el caso anterior, equivale a [^0-9]:

Caracteres especiales: Character classes.

/\s/ Whitespace nos permite capturar un espacio en blanco:



/\S/ Not whitespace al revés que el caso anterior:

Caracteres especiales: Character classes.

/\t/ Tab nos permite capturar una tabulación:

/\n/ Line feed nos permite capturar un salto de línea:

```
/\n/.test("salto \n de línea")
true
```

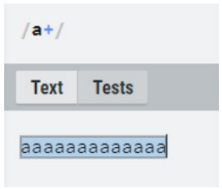
Caracteres especiales: Character classes.

/./ Dot nos permite capturar cualquier caracter:

Caracteres especiales: Quantifiers.

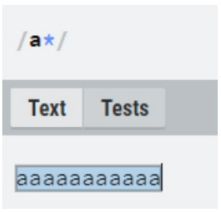
/a?/ Optional el elemento o grupo que precede a este caracter es opcional puede estar o no:

/a+/ Plus el elemento o grupo que precede a este caracter aparecerá una o más veces:



Caracteres especiales: Quantifiers.

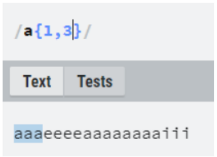
/a*/ Star el elemento o grupo que precede a este caracter aparecerá cero o más veces:



/a{4}/ Quantifier delimita el número del elemento o grupo que precede a este cuantificador:

Caracteres especiales: Quantifiers.

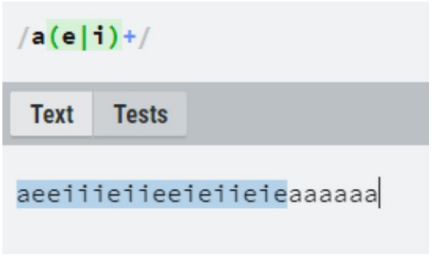
/a{1,3}/ Quantifier delimita el número del elemento o grupo que precede a este cuantificador entre un valor mínimo y un valor máximo:



/a{1,}/ Quantifier delimita el número del elemento o grupo que precede a este cuantificador con un valor mínimo pero un máximo indefinido:

Caracteres especiales: Quantifiers.

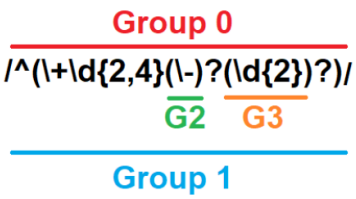
/a(e|i)+/ Alternation finalmente el alternador que nos permite elegir entre múltiples posibilidades:

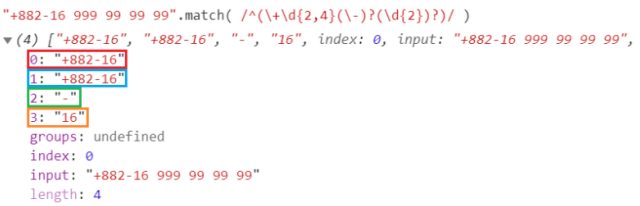


Groups & ranges

En el ejemplo anterior vimos un paréntesis con dos posibles valores, **los paréntesis son grupos de captura**.

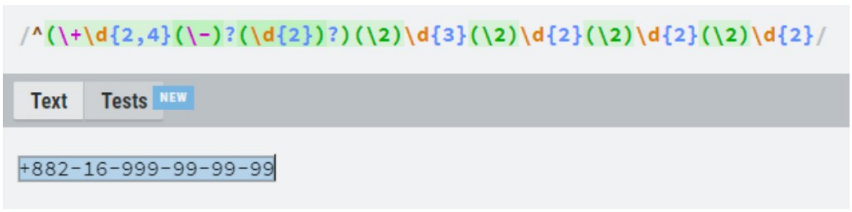
Ahora mismo lo más **importante** es saber que los grupos de captura se clasifican en orden de aparición, priorizando los paréntesis internos y posteriormente los contiguos. En el que, aunque no usemos paréntesis, por defecto el grupo de captura [0] será toda la cadena coincidente con nuestra RegExp.

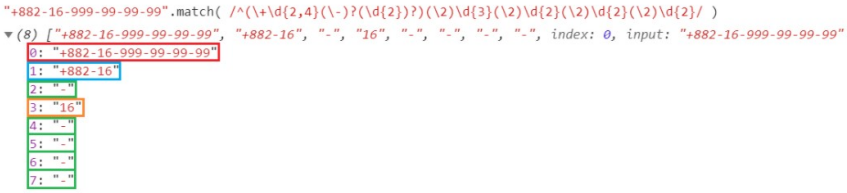




Groups & ranges

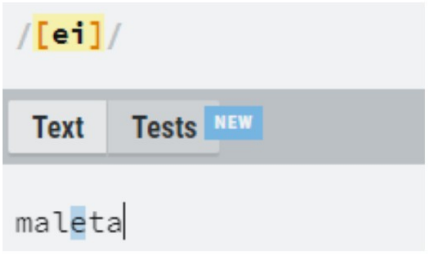
/(\n)/ Numeric reference en razón al orden de los grupos, podemos referenciar el valor de un grupo para que se repita su comprobación en el patrón:





Groups & ranges

/[ei]/ Character set nos permite seleccionar un caracter que coincidan con algún elemento entre ambos corchetes:

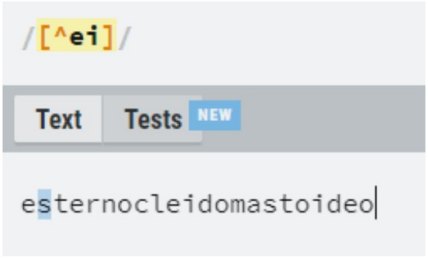


/[ei]/g Character set usando la globalidad nos permite seleccionar aquellos caracteres que coincidan con algún elemento entre ambos corchetes:

Groups & ranges

/[^ei]/g Negated set usando la globalidad nos permite seleccionar aquellos caracteres que no pertenezcan a alguno de los elementos entre ambos corchetes:

/[^ei]/ Negated set nos permite seleccionar un caracter que no coincidan con algún elemento entre ambos corchetes:



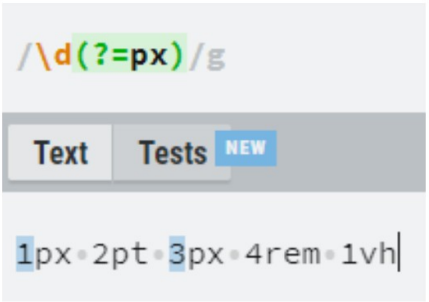
Groups & ranges

/[A-Z]/g Range usando la globalidad podemos seleccionar todos aquellos caracteres que están comprendidos entre la "A" y "Z" en el código ASCII:

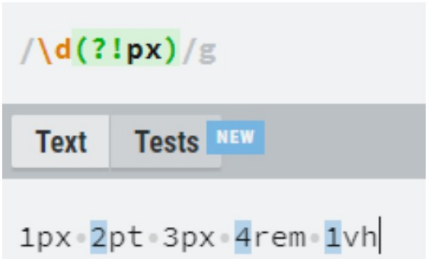
/[^A-Z]/ Negated range selecciona todos los caracteres que no estén comprendidos en dicho rango:

Lookaround

/\d(?=px)/g Positive lookahead usando la globalidad podemos seleccionar todos aquellos dígitos que haya delante de una cadena "px":

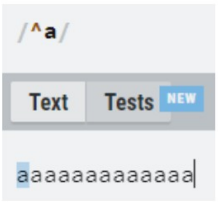


/\d(?!px)/g Negative lookahead usando la globalidad podemos seleccionar todos aquellos dígitos que **no** haya delante de una cadena "px":

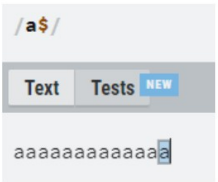


Anchors

/^a/ Beginning la cadena empieza por... :

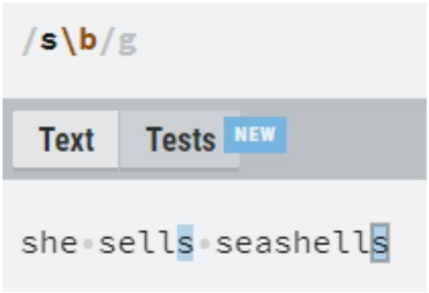


/a$/ End la cadena termina en... :

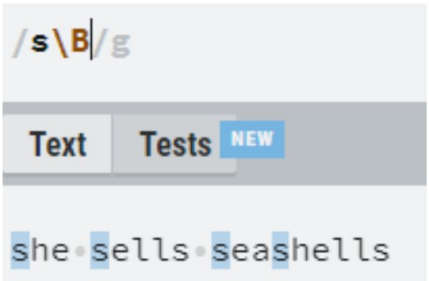


Anchors

/s\b/g Word boundary usando la globalidad selecciona todas las letras "s" del final de cada palabra:



/s\B/g Not word boundary usando la globalidad selecciona todas las letras "s" que no estén al final de cada palabra:



Escaped characters

/\(\)\[\]\.\+*\?\-\\/ **Escaped characters** podemos comprobar si la cadena posee alguno de los caracteres que tienen un comportamiento predeterminado en un RegExp (por ejemplo "?") empleando la barra invertida delante de dicho caracter:

Flags

/a/ De esta forma solo accedemos a la primera letra "a" minúscula que se encuentre:

/a/g Global search: si usamos la letra "g" justo tras el cierre del patrón buscará más de una coincidencia con la letra "a" minúscula:

Flags

/a/ De esta forma solo accedemos a la primera letra "a" minúscula que se encuentre:

/a/i Ignore case: De esta forma solo accedemos a la primera letra "a" que se encuentre tanto si es minúscula como si no lo es:

/a/ig Ignore case + global search: De esta forma accederemos a todas las letras "a" tanto minúsculas como mayúsculas:

test() y exec()

El método **test()** nos permite verificar que una cadena cumple con el patrón que realiza el test:

El método **exec()** nos devuelve un array con el elemento encontrado y el índice donde acaba dentro de la cadena. Para ello debemos iterar el resultado y **es obligatorio usar el global search**:

match() y matchAll()

El método **match()** nos devuelve un array con los grupos seleccionados (por defecto el grupo [0] que es toda la cadena coincidente con el RegExp). Para usar match debemos evitar la flag de la búsqueda global:

El método **matchAll()** nos devuelve un array formado por otro arrays con las coincidencias que haya encontrado en la cadena, es decir, es un array que agrupa los arrays de varios match, uno por cada coincidencia. En este caso **es obligatorio usar la flag de la búsqueda global**:

replace(), search() y split()

El método **replace()** devuelve una nueva cadena en la que se ha reemplazado todas aquellas coincidencias con el patrón por la cadena pasada por segundo parámetro:

El método **search()** nos devuelve el índice donde se ubica el inicio de la coincidencia, en caso de no encontrar ninguna devolvería -1:

El método **split()** nos devuelve un array formado por aquellas secciones de la cadena original resultantes de haberla troceado por los elementos coincidentes con el patrón: