UT6. Expresiones Regulares

EXPRESIONES REGULARES

Se utilizan, sobre todo, para establecer un patrón que permita establecer condiciones avanzadas en los textos, de modo que podamos validar los textos que encajen con ese patrón. Las labores típicas en las que ayudan las expresiones son: validación de errores, búsqueda de textos con reglas complejas y modificación avanzada de textos.
Las expresiones regulares de JavaScript en realidad son objetos de tipo **RegExp**.

Las expresiones regulares se pueden crear indicando las mismas entre dos barras de dividir. Entre ambas barras se coloca la expresión regular. Tras las barras se pueden indicar modificadores, la sintaxis sería esta:

/expresionRegular/

Así, para crear un objeto que contenga una expresión regular podemos hacer lo siguiente:
let expNIF=/[KLXYZ0-9][0-9]{7}[A-Z]/; //establece el patrón que cumple el NIF.

Otra forma de crear la expresión, usando una notación más formal es:
let expNIF=new RegExp('[KLXYZ0-9][0-9]{7}[A-Z]');

ELEMENTOS DE LAS EXPRESIONES REGULARES
El patrón de una expresión regular puede contener diversos símbolos que se interpretan de forma especial. Para comprobar si un texto cumple con el patrón de una expresión regular, disponemos del método **test**. Ejemplo:
let expresion=/c/;
console.log(expresion.test('casa')); /Escribe true
console.log(expresion.test('Casa')); //Escribe false
console.log(expresion.test('barbacoa')); //Escribe true

Si queremos no distinguir entre mayúsculas y minúsculas, podemos añadir el modificador i al final de la expresión. La letra i minúscula tras el cierre de la expresión indica que no se debe distinguir entre mayúsculas y minúsculas:
let expresion=/c/i;
console.log(expresion.test('casa')); /Escribe true
console.log(expresion.test('Casa')); //Escribe true

Nota: Se puede utilizar la letra "g" tras la expresión regular para que realice la verificación de forma global y no se pare tras encontrar una correspondencia.
let expresion=/c/g;
let expresion = new RegExp('c','g'); //con el objeto RegExp se pasa como parámetro

Si en lugar de un carácter usamos varios, solo los textos que contengan esos caracteres en ese orden cumplirán el patrón.
let expresion=/as/;
console.log(expresion.test('casa')); //Escribe true
console.log(expresion.test('Casa')); //Escribe true
console.log(expresion.test('asador')); //Escribe true
console.log(expresion.test('valiosa')); //Escribe false

Que comience por

El símbolo circunflejo (^) obliga a que el siguiente símbolo de la expresión sea el primero que, obligatoriamente, tenga el texto que validemos:

let expresion=/^c/;
console.log(expresion.test('casa')); //Escribe true
console.log(expresion.test('barbacoa')); *//Escribe false*

Que termine por

El símbolo de dólar ($) hace que el carácter anterior sea, obligatoriamente, el último del texto.

let expresion=/a$/;
console.log(expresion.test('casa')); //Escribe true
console.log(expresion.test('barbacoa')); //Escribe true
console.log(expresion.test('río')); //Escribe false, no acaba por "a"

Comodín de cualquier carácter

El punto (.) es un símbolo especial de las expresiones regulares que representa un carácter cualquiera. Ejemplo:

let expresion=/a.e/;
console.log(expresion.test('caserío')); //Escribe true
console.log(expresion.test('aereo')); //Escribe false, debería haber un carácter entre "a" y "e"
console.log(expresion.test('alambique')); //Escribe false

Símbolos opcionales

Cuando se pone una serie de símbolos entre corchetes, se permite cualquier de ellos. Por ejemplo:

let expresion=/[ao]$/; //Es válido cualquier texto que acabe por "a" u "o"
console.log(expresion.test('casa')); //Escribe true
console.log(expresion.test('barbacoa')); //Escribe true
console.log(expresion.test('río')); //Escribe true

Podemos incluso indicar rangos. Por ejemplo, ***la expresión [a-z]*** significa cualquier carácter entre la letra a y la z minúscula.

let expresion=/^[a-z]/; //Es válido cualquier texto que empiece con minúsculas
console.log(expresion.test('casa')); //Escribe true
console.log(expresion.test('Casa')); //Escribe false
console.log(expresion.test('río')); //Escribe true
console.log(expresion.test ('árbol')); //Escribe false

Puede sorprendernos que se dé como falso el texto **árbol**, pero es que el carácter á en Unicode no está entre la letra a y la z, está mucho después. Por eso hay que usar estas expresiones con cautela. Por ejemplo podríamos cambiar el código por:

let expresion=/^[a-záéíóúü]/; //Es válido cualquier texto que empiece con minúsculas
console.log(expresion.test('casa')); //Escribe true
console.log(expresion.test('Casa')); //Escribe false
console.log(expresion.test('río')); //Escribe true
console.log(expresion.test('árbol')); //Escribe true

Carácter no permitido

Si dentro del corchete indicamos un carácter circunflejo, estamos indicando que el contenido no se tiene que cumplir. Por ejemplo: [^ae] significa que no puede aparecer ni la letra a ni la e, cualquier otra sería válida. Ejemplo:

let expresion=/^[^AEIOU]/; // Que no empiece por ninguna vocal en mayúsculas.
console.log(expresion.test('Empresa')); //Escribe false
console.log(expresion.test('Casa')); //Escribe true
console.log(expresion.test('Ala')); //Escribe false

Símbolos de cardinalidad

Son símbolos que sirven para indicar la repetición de una expresión. Son, concretamente:

SÍMBOLO	SIGNIFICADO
x+	La expresión a la izquierda de este símbolo se puede repetir una o más veces
X'*'	la expresión a la izquierda de este símbolo se puede repetir cero o más veces
X?	El carácter a la izquierda de este símbolo se puede repetir cero o una veces
x{n}	Significa que la expresión ***x*** aparecerá ***n*** veces, siendo n un número entero positivo.
x{n,}	Significa que la expresión ***x*** aparecerá ***n*** o más veces.
x{m,n)	Significa que la expresión ***x*** aparecerá de ***m*** a ***n*** veces.

Ejemplos:

let expresion=/^[AEIOU].+a/;
console.log(expresion.test("Asa")); //Escribe true
console.log(expresion.test("Estación")); //Escribe true
console.log(expresion.test("Ea")); //Escribe false
La última expresión es falsa porque debería haber un símbolo entre la E y la a. En cambio:

let expresion=/^[AEIOU].*a/;
console.log(expresion.test("Ea")); //Escribe true

Si quisiéramos validar un código postal español, debemos indicar que solo se permiten 5 caracteres numéricos, que tienen que ser eso, exactamente cinco y, por ello, se indican los delimitadores de principio y final.

let CPvalido=/^[0-9]{5}$/;

Paréntesis

Los paréntesis permiten agrupar expresiones. Eso permite realizar expresiones aún más complejas. Ejemplo:

let expresion=/([a-z]{2}[0-9]){3}/;
Eso obliga a que el texto tenga dos letras de la **a** a la **z** en un texto, luego un número. El último número entre llaves hace referencia a todo el paréntesis, por lo que las dos letras y el número deberán aparecer tres veces seguidas. Uso posible:

let expresion=/([a-z]{2}[0-9]){3}/;
console.log(expresion.test("ad3rf1hj4")); //Escribe true
console.log(expresion.test("a3f1j4")); //Escribe false
console.log(expresion.test("ab3fg1")); //Escribe false

Opciones

La barra vertical "|" permite indicar que se da por válida la expresión de la izquierda o la de la derecha. Ejemplo:

let expresion=/^([A-Z][0-9]{6})|([0-9][A-Z]{6})$/;
console.log(expresion.test("A123456")); //Escribe true
console.log(expresion.test("1ABCDEF")); //Escribe true

Ejemplo de expresión que valida un código postal (formado por 5 números del 00000 al 52999). Los paréntesis ayudan a agrupar esta expresión:

let cpl="49345";
let cp2="53345";
let expresion=/^((5[012]) \| ([0-4][0-9])) ([0-9]{3})$/;
console.log(exp.test(cpl)); //Escribe true
console.log(exp.test(cp2)); //Escribe false

Símbolos abreviados

Son símbolos que se usan con el carácter backslash y que permite escribir expresiones de forma aún más rápida. Además funcionan muy bien con Unicode:

SÍMBOLO	SIGNIFICADO
\d	Dígito, vale cualquier dígito numérico.
\D	Cualquier carácter salvo los dígitos numéricos
\s	Espacio en blanco.
\s	Cualquier carácter salvo el espacio en blanco
\w	Vale cualquier carácter alfanumérico ***(word).*** Es lo mismo que [a-zA-Z0-9]
\W	Cualquier carácter que no sea alfanumérico. Equivalente a [^A-Za-z0-9]
\0	Carácter nulo.
\n	Carácter de nueva línea.
\t	Carácter fabulador.
****	El propio símbolo \
\"	Comillas dobles.
V	Comillas simples.
\c	Permite representar el carácter c cuando este sea un carácter que de otra manera no sea representable (como [,], /, \,...). Por ejemplo \\ es la forma de representar la propia barra invertida.
\ooo	Permite indicar un carácter Unicode mediante su código octal.
\xflf	Permite indicar un carácter ASC11 mediante su código hexadecimal.
\uflfff	Permite indicar un carácter Unicode mediante su código hexadecimal.

MÉTODO EXEC

Este método es similar a test en cuanto a que valida una expresión regular en un texto. Pero lo que hace es devolver un array con información sobre cuándo se encontró la expresión en el texto. En este array los elementos son:

- Elemento con índice 0. Es el primer texto encontrado que cumple la expresión.
- Elemento con propiedad Índex. Es un número que indica la primera posición en la que se encontró el texto.
- Elemento con propiedad input. Es el texto original.
- Se pueden usar subcadenas de búsqueda entre paréntesis y entonces los índices 1,2,.... nos devolverían los textos encontrados con las subcadenas.

Como vemos, el objeto devuelto es complejo. Ejemplo:

let texto="Este es un texto de prueba";
let expresion=/\w+/;
let res=expresion.exec(texto);
console.log(res);

Resultado:

```
[ 'Este',
  índex: 0,
  input: 'Este es un texto de prueba',
  groups: undefmed ]
```

El primer elemento (res [0]) muestra el primer texto que cumple la expresión, la cual busca una serie de letras. Por eso es la palabra ***Este*** la primera que lo cumple (después viene un espacio en blanco que ya no es una letra). El elemento siguiente: res.index o también: res["index"], indica el índice del texto en el que empieza el texto que cumple esa expresión. Finalmente res.input escribe el texto tal cual.