APRENDERAPROGRAMAR.COM

PARÉNTESIS EN EXPRESIONES REGULARES JAVASCRIPT. FLAGS. MÉTODOS EXEC, TEST, MATCH, SEARCH, REPLACE, SPLIT. EJEMPLOS (CU01155E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº55 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

PARÉNTESIS Y FLAGS EN EXPRESIONES REGULARES

Las expresiones regulares JavaScript son una potente herramienta que nos permitirá resolver problemas diversos. Ya conocemos aspectos básicos como la instanciación de objetos RegExp y los caracteres especiales disponibles. Vamos a ampliar conocimientos sobre expresiones regulares.

EXPRESIONES REGULARES CON FLAGS

JavaScript permite especificar modos de búsqueda de expresiones regulares a través de lo que denomina flags (banderas o indicadores).

Los flags se indican en el momento de definir la expresión regular:

a) Si se define en forma de literal: var miExpresionRegular = /as?.a/**flagsAIncluir**

b) Instanciando el objeto RegExp: var miExpresionRegular = new RegExp("as?.a", " **flagsAIncluir**")

En la siguiente tabla se indican los principales flags, cómo usarlos y ejemplos.

Flag	Significado	Ejemplo sintaxis
g	Búsqueda global de todas las coincidencias (no se detiene al encontrar la primera coincidencia)	/as?.a/**g** new RegExp("as?.a", "**g**")
i	No diferenciar entre mayúsculas y minúsculas	/as?.a/**i** new RegExp("as?.a", "**i**")
m	Búsqueda multilínea. Para los indicadores de comienzo de cadena ^ y fin de cadena $, si se aplica esta bandera, se tienen en cuenta matches en todos los principios y finales de línea, no sólo al principio y final de cadena.	/as?.a/**m** new RegExp("as?.a", "**m**")
Combinaciones	Se pueden especificar varios flags	var miExpReg = /^i/**mi** Indica que hará match con cualquier comienzo de línea por una i minúscula ó I mayúscula (búsqueda multilínea y sin diferenciar mayúsculas y minúsculas).

MÉTODOS DE LOS OBJETOS TIPO EXPRESIÓN REGULAR

Los objetos de tipo expresión regular proveen de una serie de métodos útiles. En la siguiente tabla vemos cuáles son, su significado y un ejemplo de uso.

Método	Significado	Ejemplo de uso

exec	Ejecuta una búsqueda del patrón y devuelve un array cuyo índice 0 contiene el valor encontrado. Si está activado el flag g, la repetición de la búsqueda devuelve la siguiente coincidencia.	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = '';
		while ((matches = miExpReg.exec(txt)) !== null) {
		msg = msg + 'Encontrado: '+matches[0]+'\n'; }
		alert(msg);

test	Comprueba si se verifica el patrón y devuelve true o false	var miExpReg = /p.sto/g var txt = 'el pasto es pisto pero no pesto'; msg = ''; alert('Hay match: '+ miExpReg.test(txt));

MÉTODOS DE LOS OBJETOS STRING RELACIONADOS CON EXPRESIONES REGULARES

Los objetos de tipo String proveen de una serie de métodos útiles. En la siguiente tabla vemos cuáles son, su significado y un ejemplo de uso.

Método	Signficado	Ejemplo de uso

match	Devuelve un array con las coincidencias encontradas, o null si no hay coincidencias.	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = ''; var matches = txt.match(miExpReg);
		if (matches != null) {
		for (var i=0; i<matches.length; i++){
		msg = msg + 'Encontrado: '+matches[i]+'\n'; }
		} else {msg = 'No se encontraron coincidencias';}
		alert(msg);

search	Devuelve la posición de comienzo de la primera coincidencia dentro del string, o -1 si no hay coincidencia. Recordar que la primera posición posible es cero.	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = ''; var posicion = txt.search(miExpReg);
		if (posicion != -1) {
		msg = msg + 'Encontrado patrón en posición: '+posicion+'\n';
		} else {msg = 'No se encontraron coincidencias';}
		alert(msg);

replace	Devuelve un nuevo String (sin modificar el original) donde se reemplaza una o varias coincidencias por lo especificado (una cadena o una función que devuelve la cadena).	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = ''; var nuevoTxt = txt.replace(miExpReg, 'coco');
		msg = 'txt vale ' + txt + ' y nuevoTxt vale ' +nuevoTxt;
		alert(msg);

Método	Signficado	Ejemplo de uso

split	Devuelve un array con las subcadenas resultantes de dividir la cadena original en subcadenas delimitadas por el carácter separador especificado (que queda excluido). Si se indican unas comillas vacías se extraen todos los caracteres a un array.	var miExpReg = /\d/g
		var txt = 'un 3 de bastos gana a un 5 de copas pero no a un 7 de oros';
		msg = ''; var splits = txt.split(miExpReg);
		msg = 'splits contiene ' + splits;
		alert(msg);

USAR PARÉNTESIS PARA AGRUPAR FRAGMENTOS

Los paréntesis sirven para agrupar varios caracteres y afectarlos por un carácter especial.

Por ejemplo var miExpReg = /(p.s)?to/g

var txt = 'el pasto es pisto y eso es todo';

var resultado = txt.match(miExpReg);

Hace que resultado contenga ['pasto ', 'pisto ', 'to '] ya que hay un fragmento de la expresión que se ha agrupado e indicado que es opcional.

Los paréntesis nos permiten no tener que repetir. Por ejemplo en vez de /estereotipo|estereoforma/ podemos escribir /estereo(tipo|forma).

USAR PARÉNTESIS PARA IDENTIFICAR SUBEXPRESIONES CON MATCH

Los paréntesis generan otro efecto. Cuando se invoca la función match buscando la primera coincidencia (es decir, búsqueda no generalizada sin el flag g) y existen subexpresiones entre paréntesis, el array generado guarda en su índice cero la primera coincidencia mientras que en los sucesivos índices guarda las coincidencias de las subexpresiones.

Cuando se escribe una expresión regular, los elementos que se incluyen entre paréntesis pueden ser recuperados por separado invocándolos luego como $1, $2, $3…. Estas variables persisten normalmente hasta que se invoca una nueva expresión regular.

Veámoslo con un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var expReg = /(\w+)\s(\w+)\s(\w+)/;
```

```
var txt = "Era una noche negra y sola";
var newTxt = txt.match(expReg);
console.log(newTxt); //Activar la consola para que se visualice
var cambiada = newTxt[0].replace(newTxt[3], 'madrugada');
console.log(cambiada); //Activar la consola para que se visualice
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es:

Array ["Era una noche", "Era", "una", "noche"]

"Era una madrugada"

Analicemos lo que hace el código: en el array newTxt queda almacenado en su posición cero la coincidencia global con la expresión regular. En su índice uno, queda almacenado el match de la primera subexpresión entre paréntesis, es decir, la primera palabra (sin tildes). En su índice dos, la segunda palabra, y en su índice 3, la tercera palabra. A continuación creamos una cadena formada por las tres palabras con la tercera de ellas reemplazada por 'madrugada'.

USAR PARÉNTESIS PARA IDENTIFICAR SUBEXPRESIONES CON REPLACE

Dentro de una sentencia replace, los matches de subexpresiones entre paréntesis pueden ser invocadas usando unas variables temporales especiales que se numeran como $1, $2, $3... etc. de acuerdo con la cantidad de subexpresiones entre paréntesis que existan. Este ejemplo nos muestra cómo se pueden usar:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var expReg = /(\w+)\s(\w+)\s(\w+)/;
var txt = "Alberto Flores Rubalcaba: tiene calificación de 10";
var cambiada = txt.replace(expReg, '$3 $2, $1');
console.log(cambiada); //Activar la consola para que se visualice
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

En este ejemplo el match de la expresión regular es <<Alberto Flores Rubalcaba>>, y cada palabra es una subexpresión por encontrarse entre paréntesis en la expresión regular. En el replace podemos identificar cada subexpresión como $1 (primera subexpresión), $2 (segunda) y $3 (tercera).

En el replace indicamos que la coincidencia de la expresión regular sea sustituida por la cadena formada por la tercera palabra seguida de la segunda, una coma y la primera palabra.

De este modo el resultado esperado es que se muestre por consola: "Rubalcaba Flores, Alberto: tiene calificación de 10".

EJERCICIO

Crea un documento HTML (página web) donde exista un formulario que se envíe por el método GET. Se pedirá al usuario que introduzca nombre, apellidos y correo electrónico. Define dentro de la etiqueta form que cuando se produzca el evento onsubmit (pulsación del botón de envío del formulario) se ejecute una función a la que denominaremos validacionConExpReg que no recibe parámetros.

La función validar debe realizar estas tareas y comprobaciones **utilizando expresiones regulares**:

a) Comprobar que el nombre contiene al menos tres letras. Si no es así, debe aparecer un mensaje por pantalla indicando que el texto no cumple tener al menos tres letras. Por ejemplo si se trata de enviar Ka como nombre debe aparecer el mensaje: “El nombre no cumple tener al menos tres letras”.

b) Comprobar que el correo electrónico contiene el carácter @ (arroba) y el carácter . (punto). De no ser así, deberá aparecer un mensaje indicando que al correo electrónico le falta uno o ambos caracteres. Por ejemplo si se trata de enviar pacogmail.com deberá aparecer el mensaje: “Falta el símbolo @ en el correo electrónico”.

c) Antes de enviarse los datos del formulario a la página de destino, todas las letras del correo electrónico deben transformarse a minúsculas. Por ejemplo si se ha escrito PACO@GMAIL.COM debe enviarse paco@gmail.com

d) Antes de enviarse los datos del formulario a la página de destino, si el correo electrónico contiene la subcadena “ at “ debe reemplazarse por el símbolo @. Por ejemplo si se ha escrito paco at gmail.com debe enviarse paco@gmail.com

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01156E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206