JavaScript en el navegador - MÓDULO 6:

Strings, iteradores, bucles y Regexp

Juan Quemada, DIT - UPM

Índice

Strings, códigos, UNICODE, literales, plantillas y códigos escapados

Juan Quemada, DIT - UPM

Alfabeto, código y codificación

- **Juego de caracteres** o **alfabeto**

  - **Conjunto de símbolos** normalizados para representar una lengua

- **Código de caracteres**
  - Conjunto de **puntos de código** dados a los símbolos de un alfabeto, p.e.
    - **ASCII:** alfabeto inglés codificado en 7 bits (128 caracteres y 95 imprimibles)
    - **ISO-8859-1, 2, .., 15:** Alfabetos de Europa occidental codificados en 8 bits
    - **UNICODE:** código internacionalizado que contiene casi todos los alfabetos
      - Posee 17 planos codificados en 2 bytes cada uno (1er Plano: BMP)

Char	Code
@	80
A	81
B	82
C	83

- **Codificación**
  - **representación binaria** de un código de caracteres

    - **ASCII e ISO Latin-x**: el valor del **punto del código** coincide con la codificación binaria
    - **UNICODE UTF-8:** Codificación binaria en 1, 2, 3 o 4 bytes, eficiente con lenguas latinas
    - **UNICODE UTF-16:** Codificación del plano BMP en 2 bytes y de otros planos en 4 bytes
    - **UNICODE UTF-32:** Codificación de todos los planos en 4 bytes

El tipo string

```
"Son " + 4*2  + " coches" =>  "Son 8 coches"
`Son ` + 4*2 + 'coches'   =>  "Son 8 coches"
`Son ${2*4} coches`       =>  "Son 8 coches"
```

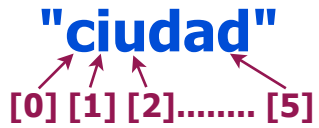
- **Literales de string**
  - Se delimitan por **comillas** o **apóstrofes: "hola, que tal"** u **'hola, que tal'**
    - Ejemplo de "texto 'entrecomillado' "
  - **Plantilla de string** (ES6): delimitada por **comillas invertidas `...`**
    - Son strings multi-línea y con expresiones (**${expr}**) que se evalúan y sustituyen
      - Por ejemplo: **`Un día tiene ${24*60} minutos y ${24*60*60} segundos`**
    - La plantilla de string se denomina a veces interpolación o interpolador de string.
- **Los strings están basados en UNICODE y soportan muchos alfabetos**
  - **"hola que tal" en griego y chino: 'Γεια σου, ίσως' o '嗨，你好吗'**
    - Alfabetos de UNICODE: http://www.unicode.org/charts/
- Operador de **concatenación de strings**: **+**
  - **'Hola' + " " + `Pepe` => "Hola Pepe"**
- **String(a)** (función de conversión a string)
  - El método **toString()** de cada clase transforma el objeto a string al realizar la conversión

```
String(-4)          => "-4"
String(NaN)         => "NaN"
String(Infinity)    => "Infinity"
String(undefined)   => "undefined"
```

```
(-4).toString()          => "-4"
(NaN).toString()         => "NaN"
(Infinity).toString()    => "Infinity"
(undefined).toString()   => "undefined"
```

Mas info: https://javascript.info/string

string: array y clase




```
'ciudad'[2]        => 'u'
'ciudad'.length    => 6
```

- Un **string** es un **array** de caracteres
  - Se pueden acceder con un índice

- Clase **String**:
  - Define métodos para strings, tales como substring(..), charCodeAt(..), indexOf(..), ..
    - Doc: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

```
// ¿Contiene el substring indicado?
'aananan'.includes("na")      =>  true

// Índice del substring (1a instancia)
'aananan'.indexOf("an")       =>  1

// Índice del substring desde índice 2
'aananan'.indexOf("an", 2)    =>  3

// Índice de la última instancia del substring
'aananan'.lastIndexOf("an")   =>  5
```

```
// Substring entre índices 2, 5
'ciudad'.substring(2,5)           =>  'uda'

// Código UNICODE del carácter de índice
'ciudad'.charCodeAt(2)            =>  117

// carácter asociado a código UNICODE
String.fromCharCode(117)      =>  'u'
```

Códigos escapados

- **Códigos escapados**: comienzan por ****
  - Permiten representar caracteres o símbolos no incluidos en el teclado, por ejemplo
    - El ideograma chino **"丰"** que es **"\u2F80"**
    - El símbolo matemático **"∑"** que es **"\u2211"**
  - O caracteres especiales o de control: **"\n"**
- Caracteres especiales **ASCII**, tales como
  - "Comillas dentro de **\"comillas\""** deben escaparse
  - Nueva línea: "Dos **\n** lineas", **\n** separa líneas.
- Caracteres **ISO-8859-1**
  - Código **\xHH** (**HH:** punto de código en hex)
    - Por ejemplo, "Dos **\x0A** lineas" equivale a "Dos **\n** lineas"**.**
- Caracteres **UNICODE**
  - Código **\uHHHH** (**HHHH:** punto de código en hex)
    - Por ejemplo, "Dos **\u000A** lineas" equivale a "Dos **\n** lineas"**.**

Caracteres escapados ASCII, ISO-8859-1 y UNICODE

NUL (nulo):	**\0, \x00, \u0000**
Backspace:	**\b, \x08, \u0008**
Horizontal tab:	**\t, \x09, \u0009**
Newline:	**\n, \x0A, \u000A**
Vertical tab:	**\v, \x0B, \u000B**
Form feed:	**\f, \x0C, \u000C**
Carriage return:	**\r, \x0D, \u000D**
Comillas (dobles):	**\", \x22, \u0022**
Apóstrofe :	**\', \x27, \u0027**
Backslash:	**\\, \x5C, \u005C**

Caracteres ASCII (Basic Latin) en UNICODE BMP

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077

8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

Extensión ISO Latin1 en UNICODE BMP

¥ será "\xA5" o "\u00A5"

© UNICODE: http://www.unicode.org/charts/PDF/U0080.pdf

	008	009	00A	00B	00C	00D	00E	00F
0	XXX 0080	DCS 0090	NB SP 00A0	° 00B0	À 00C0	Đ 00D0	à 00E0	ð 00F0
1	XXX 0081	PU1 0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	BPH 0082	PU2 0092	¢ 00A2	2 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	NBH 0083	STS 0093	£ 00A3	3 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	NEL 0085	MW 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HTJ 0089	XXX 0099	© 00A9	1 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	VTS 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	SS3 008F	APC 009F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

Radicales Kangxi de UNICODE BMP

© UNICODE: http://www.unicode.org/charts/PDF/U2F00.pdf

鬲 será "\u2FC0"

	2F0	2F1	2F2	2F3	2F4	2F5	2F6	2F7	2F8	2F9	2FA	2FB	2FC	2FD
0	一 2F00	凵 2F10	士 2F20	己 2F30	支 2F40	比 2F50	瓜 2F60	示 2F70	聿 2F80	衣 2F90	辰 2FA0	革 2FB0	鬲 2FC0	鼻 2FD0
1	丨 2F01	刀 2F11	夕 2F21	巾 2F31	攴 2F41	毛 2F51	瓦 2F61	内 2F71	肉 2F81	襾 2F91	辵 2FA1	韋 2FB1	鬼 2FC1	齊 2FD1
2	丶 2F02	力 2F12	夂 2F22	干 2F32	文 2F42	氏 2F52	甘 2F62	禾 2F72	臣 2F82	見 2F92	邑 2FA2	韭 2FB2	魚 2FC2	齒 2FD2
3	丿 2F03	勹 2F13	夕 2F23	幺 2F33	斗 2F43	气 2F53	生 2F63	穴 2F73	自 2F83	角 2F93	酉 2FA3	音 2FB3	鳥 2FC3	龍 2FD3
4	乙 2F04	匕 2F14	大 2F24	广 2F34	斤 2F44	水 2F54	用 2F64	立 2F74	至 2F84	言 2F94	釆 2FA4	頁 2FB4	鹵 2FC4	龜 2FD4
5	亅 2F05	匚 2F15	女 2F25	廴 2F35	方 2F45	火 2F55	田 2F65	竹 2F75	臼 2F85	谷 2F95	里 2FA5	風 2FB5	鹿 2FC5	龠 2FD5

Iteradores y bucles:

forEach, find-index, filter, map, reduce, for-in, for-of, keys, values y entries

Juan Quemada, DIT - UPM

Métodos iteradores: forEach

- **Objeto iterable**
  - Objeto compuesto cuyos componentes puede procesarse, de uno en uno, con una función
    - Por ej, un iterador puede extraer y procesar los **elementos** de un **arrays,** de uno en uno, desde **0** a **length-1**
- Método iterador **forEach**
  - Ejecuta una función para cada elemento de un objeto iterable, p.e un **array**
- **forEach(<función>)** invoca **<función>** para cada elemento
  - forEach(**function(elem, i, a, _this)** {…}) o forEach(**(elem, i, a, _this) => {…}**)
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach
- Los iteradores equivalen a **bucles** en programación funcional
  - Ejecutan la función (parámetro) en cada elemento de un array (u objeto iterable)
    - Estos dos ejemplos son equivalentes: **ambos** incluyen un **bucle** que **suma** los **elementos** de un **array**

```
let n = [7, 4, 1, 23];
let add = 0;

for (let i=0; i < n.length; ++i){
   add += n[i];
}

add          // => 35   (7+4+2+23)
```

```
let n = [7, 4, 1, 23];
let add = 0;

n.forEach(elem => add += elem)

add          // => 35
```

Otros métodos iteradores de Array

- Estos métodos invocan la función también con los mismos 3 parámetros
  - **elem**: elemento del array accesible en la invocación en curso
  - **i**: índice al elemento del array accesible en la invocación en curso
  - **a**: array completo sobre el que se invoca el método

- **find**((elem, i, a) => {…})

```
[7, 4, 1, 23].find(elem => elem < 3);          //  =>   1
```

  - Retorna el 1er elemento donde la función retorna true
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

- **findIndex**((elem, i, a) => {…})

```
[7, 4, 1, 23].findIndex(elem => elem < 3);   //  =>   2
```

  - Retorna el índice del 1er elem. donde la función retorna true
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex

- **filter**((elem, i, a) => {…})

```
[7, 4, 1, 23].filter(elem => elem > 5);          //  =>   [7, 23]
```

  - Retorna un nuevo array sin los elementos para los que la función retorna false
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

- **map**((elem, i, a) => {…})

```
[7, 4, 1, 23].map(elem => -elem);       //  =>   [-7, -4, -1, -23]
```

  - Retorna un nuevo array sustituyendo cada elemento por el que retorna la función
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Método reduce

- El método **reduce** añade un parámetro **acumulador: acc**
  - **acc**: variable con el valor retornado por la invocación anterior de la función
    - además están los 3 parámetros típicos de los métodos iteradores: **element, index y array**
- **reduce**((acc, element, index, array) => {…}, acc_initialisation)
  - Inicializa accumulator **acc** con **acc_initialisation** e itera de **0** a **array.length-1**
    - **acc** recibe en cada iteración el valor retornado por la función en la iteración anterior
      - https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/reduce
  - si **acc_initialisation** se omite, el valor inicial de acc es **array[0]** e itera de **1** a **array.length-1**

```
// Example of addition of numbers with reduce
[7, 4, 1, 23].reduce((acc, elem) =>  acc += elem, 0);      //  => 35
```

```
// Example which orders first the array and eliminates then duplicated numbers
[4, 1, 4, 1, 4].sort().reduce((ac, el, i, a)  =>  el!==a[i-1] ? ac.concat(el) : ac, []);   //  => [1, 4]
```

```
// sort(..) and reduce(..) are composed in series, where each one performs the following
[4, 1, 4, 1, 4].sort();                                                                    //  => [1, 1, 4, 4, 4]
[1, 1, 4, 4, 4].reduce((ac, el, i, a)  =>  el!==a[i-1] ? ac.concat(el) : ac, []);  //  => [1, 4]
```

Bucles for-in y for-of

- JavaScript incluye el bucle **for-in** para iterar en las propiedades de un objeto
  - ES6 añade el bucle **for-of** para iterar en los elementos de un objeto iterable: array, string, etc.
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in
- La sentencia **for-in** itera en **objetos** y en arrays
  - Al iterar en **objetos**, la variable del bucle itera con el **nombre** de la **propiedad** (string)
  - Al iterar en **arrays**, la variable del bucle itera con el **índice** del **elemento** (número)
- La sentencia **for-of** solo itera en objetos iterables, pero **no itera en objetos**
  - La variable del bucle contiene en cada iteración un **elemento** del array (u objeto iterable)
    - En un array for-of comienza por el elemento de índice **0** y termina con el de **length-1**
  - Estos 2 ejemplos de suma de elementos de Array con los nuevos bucles equivalen a los 3 ya vistos

```
let n = [7, 4, 1, 23];
let add = 0;

for (let i in n){
   add += n[i];
}

add          // => 35
```

```
let n = [7, 4, 1, 23];
let add = 0;

for (let elem of n){
   add += elem;
}

add          // => 35
```

```
let n = [7, 4, 1, 23];
let add = 0;

for (let i=0; i < n.length; ++i){
   add += n[i];
}
add          // => 35
```

```
let n = [7, 4, 1, 23];
let add = 0;

n.forEach(elem => add += elem)

add          // => 35
```

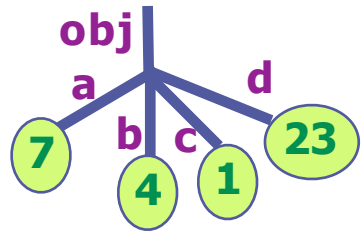
```
[7, 4, 1, 23].reduce((acc, elem) =>  acc += elem, 0);    // => 35
```

Iterar en objetos: for-in, keys, values, entries

- Sentencia **for-in**
  - Itera en las propiedades de un **objeto**, según el orden de creación de propiedades
    - **p** contiene un string con el nombre de la propiedad para acceso con notación array: **obj[p]**
  - La sentencia recorre las propiedades enumerables del objeto y de sus prototipos
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in
- Métodos estáticos **keys(..)**, **values(..)**, **entries(..)** de Object (de ES6)
  - Devuelve un **array** con las **propiedades**, **valores** o **entradas** del objeto
    - Devuelven solo las **propiedades propias** y no las de sus prototipos (conviene utilizarla)
- Por ejemplo
  - **Object.keys({a:3, b:2}) => ['a', 'b']** // devuelve array de nombres de propiedades
  - **Object.values({a:3, b:2}) => [3, 2]** // devuelve array de valores
  - **Object.entries({a:3, b:2}) => [['a',3], ['b',2]]** // devuelve array de pares nombre-valor

```
let obj = {a:7, b:4, c:1, d:23};
let add = 0;

for (let p in obj) {
      add += obj[p];
}

add    // => 35
```

```
let obj = {a:7, b:4, c:1, d:23};
let add = 0;

Object.keys(obj);     //    => [ "a", "b", "c", "d"]

Object.keys(obj).forEach(p => add += obj[p]);

add    // => 35
```



RegExp I: Búsqueda de patrones

Juan Quemada, DIT - UPM
Santiago Pavón, DIT - UPM

Expresiones regulares: RegExp

- Expresiones regulares:
  - Definen patrones que reconocen cadenas de caracteres específicas
    - Si un patrón reconoce una cadena, se dice también que casa (match)
      - » Mas info: https://javascript.info/regular-expressions
        https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

- En JavaScript se definen con la clase **RegExp** y se pueden crear con
  - Literal de RegExp: **/<expresion-regular>/**
  - Constructor de RegExp: **new RegExp("<expresion-regular>")**
    - Recibe un **string** como parámetro: debe representar una Regexp **correcta**

- Se aplican fundamentalmente con 2 métodos de **String**
  - **match(..)**: **indica** si un patrón está en un string y lo **extrae** como substring
  - **replace(..)**: devuelve un **string** con el **patrón reemplazado por un sustituto**

Búsqueda de patrones: match(..)

- **str.match(<patrón>)** método de String que busca **<patrón>** en el string **str**
  - Devuelve **truthy** o **falsy** (array con **primer substring** que casa o **null**)
    - Chrome, node y otros devuelven un array con mas parámetros (conviene utilizar solo el primero)
- Algunos patrones básicos
  - caracter: **/a/** reconoce solo el caracter "a"
  - sequencia: **/abc/** reconoce la secuencia "abc"
  - principio de string: **/^hoy/** reconoce "hoy" al principio del string
  - final de string: **/hoy$/** reconoce "hoy" al final del string
  - cualquier caracter: **/./** reconoce cualquier caracter

```
"Es hoy".match(/hoy/)        => truthy    (devuelve ['hoy'])

"Es hoy".match(/hoy$/)       => truthy    (devuelve ['hoy'], casa porque está al final])
"Es hoy".match(/^hoy/)       => falsy     (devuelve null porque 'hoy' no está al principio)
"Es hoy".match(/^..../)      => truthy    (devuelve los 4 primeros caracteres: ['Es h'])
```

Clases y rangos de caracteres

- **Clase de caracteres**: patrón con varios caracteres alternativos entre corchetes
  - Ejemplo de clase de caracteres: **/[aeiou]/** cualquier vocal (minúscula)
  - Ejemplo de clase negada: **/[^aeiou]/** no debe ser vocal (minúsc.)
  - Patrón **\s:** reconoce separadores **[\f\n\r\t\v\u00a0\u1680 .....]**
- **Rango de caracteres**: Patrón con un rango de caracteres de ASCII alternativos
  - Rango de caracteres: **/[a-z]/** rango “a-z” de letras ASCII
  - Patrón **\w**: equivale a **/[a-zA-Z0-9_]/**
  - Patrón **\d:** equivale a **/[0-9]/**

```
"canciones".match(/[aeiou]/)      =>  truthy  (devuelve ['a'])

"canciones".match(/c[aeiou]/)     =>  truthy  (devuelve ['ca'])

"canciones".match(/n[aeiou]/)     => truthy   (devuelve ['ne'])
```

Controles i, g, m

- **match()** admite controles: i, g y m
  - i: búsqueda **insensible** a mayúsculas
  - g: búsqueda **global** de todos los substrings que casan con el patrón
    - devuelve array con **todos** los substrings que **casan** (matches)
  - m: búsqueda **multilínea**, donde **^** y **$** representan principio y fin de línea

```
"canciones".match(/[aeiou]/g)       => truthy   (devuelve ['a', 'i', 'o', 'e'])
"canciones".match(/c[aeiou]/g)      => truthy   (devuelve ['ca', 'ci'])

"Hoy dice hola".match(/ho/i)        => truthy    (devuelve ['Ho'])
"Hoy dice hola".match(/ho/ig)       => truthy    (devuelve ['Ho', 'ho'])

"Hola Pepe\nHoy vás".match(/^Ho/g)        => truthy    (devuelve ['Ho'])
"Hola Pepe\nHoy vás".match(/^ho/gim)      => truthy    (devuelve ['Ho', 'Ho'])
```

RegExp II: Repetición y alternativa

Juan Quemada, DIT - UPM
Santiago Pavón, DIT - UPM

Operadores de Repetición

- **+** (una o más veces): **/a+/** casa con: “a”,“aa”, “aaa”, ..
- **?** (cero o una vez): **/a?/** casa solo con: “” y “a”
- ***** (cero o más veces): **/a*/** casa con: “”, “a”, “aa”, “aaa”, ...
- **{n}** (n veces): **/a{2}/** casa solo con: “aa”
- **{n,}** (n o más veces): **/a{2,}/** casa con: “aa”, “aaa”, “aaaa”, ...
- **{n,m}** (entre n y m veces): **/a{2,3}/** casa solo con: “aa” y “aaa”

"tiene".match(**/[aeiou]+/g**) => truthy (devuelve ['ie', 'e']) // cadenas no vacías de vocales

"tiene".match(**/[aeiou]?/g**) => truthy (devuelve ['', ‘i', 'e', '', 'e', '']) // vocal o nada

"tiene".match(**/[aeiou]*/g**) => truthy (devuelve ['','ie', '', 'e', '']) // cadenas de vocales incluyendo “”

"Había un niño.”.match(**/[a-zñáéíóú]+/ig**) => truthy (devuelve ['Había', 'un', 'niño'])
// casa con palabras en castellano: ascii extendido con ñ, á, é, í, ó, ú

Repetición ansiosa o perezosa

- Los operadores de repetición son “ansiosos” y casan con
  - la **cadena más larga posible que casa con el patrón**

- Pueden volverse “perezosos” añadiendo **“?”** detrás del operador
  - Entonces casan con la cadena más corta posible

```
"aaabb".match(/a+/)          => truthy    (devuelve ['aaa'])

"aaabb".match(/a+?/)         => truthy    (devuelve ['a'])

"ccaaccbccaa".match(/.+cc/)  => truthy    (devuelve ['ccaaccbcc'])

"ccaaccbccaa".match(/.+?cc/) => truthy    (devuelve ['ccaacc'])
```

Patrones alternativos

- **"|"** define dos patrones alternativos, por ejemplo
  - **/[a-z]+/** casa con palabras escritas con caracteres ASCII
  - **/[0-9]+/** casa con números decimales
  - **/[a-z]+|[0-9]+/** casa con palabras o números

```
"canciones".match(/ci|ca/)        => truthy    (devuelve [‘ca’])
"canciones".match(/ci|ca/g)       => truthy    (devuelve [‘ca','ci'])

“1 + 2 --> tres”.match(/[a-z]+|[0-9]+/g)    => truthy    (devuelve ['1', '2',
'tres'])
```

RegExp III:
Subpatrones y sustituciones

Juan Quemada, DIT - UPM
Santiago Pavón, DIT - UPM

Sustitución de patrones

- **str.replace(/<patrón>/, repuesto)** método de la clase string
  - sustituye en **str** el primer match de "patrón" por repuesto
- El patrón también puede tener controles i, g y m
  - i: insensible a mayúsculas
  - g: sustituye en str con todos los “match” por repuesto
  - m: multilínea, ^ y $ representan principio y fín de línea

```
"Número: 142719".replace(/1/, 'x')          => 'Número: x42719’

"Número: 142719".replace(/1/g, 'x')         => 'Número: x427x9’

"Número: 142719”.replace(/[0-9]+/, '<número>')   => 'Número: <número>'
```

Sustitución con subpatrones

- Dentro de un patrón podemos delimitar subpatrones
  - Un subpatrón se delimita con paréntesis
    - El patrón y los subpatrones pueden reutilizarse al sustituir con: "$0", "$1", ..
- Por ej., **/([0-9]+)(,[0-9]*)?/** tiene dos subpatrones **([0-9]+)** y **(,[0-9]*)**
  - $0 representa el match completo de todo el patrón
  - $1 representa el match del primer subpatrón
  - $2 el match del segundo subpatrón y así sucesivamente

```
"Número: 142,719".replace(/([0-9]+)(,[0-9]*)?/, '$1')      => 'Número: 142'

"Número: 142,719".replace(/([0-9]+)(,[0-9]*)?/, '0$2')     => 'Número: 0,719'

"Número: 142,719".replace(/([0-9]+),([0-9]*)?/, '$1.$2')   => 'Número: 142.719'
```

Final del tema