La sintaxis del lenguaje

Laura Isabel López Naves
IES Julián Marías

Convenciones de nombres

- Nombres descriptivos
  - Una mala costumbre habitual cuando se empieza en la programación, es darle un nombre muy poco descriptivo
  - A medida que el código crece, se vuelve insostenible
  - Evita nombres poco claros o inconsistentes como tmp, a, b2, variable2, etc…
- Índices y contadores
  - Cuando trabajamos en bucles for(o bucles en general)
  - El ámbito de una variable que actúa como contador (índice) es muy reducido (esa variable solo existe y afecta al interior del bucle).
  - Las variables que actúan como contador suelen nombrarse con una letra minúscula empezando desde i (de índice): i, j, k... A veces, también se usan letras como a, b, c... o la inicial minúscula de lo que representan: c para un contador, p para una posición, etc...

Convenciones de nombres

- Constantes, clases y variables
  - Las **constantes** son variables especiales que no varían su valor a lo largo del programa. Deben ir siempre en MAYÚSCULAS
  - Las **clases** son estructuras de código más complejas. Los nombres de las clases se escriben siempre CAPITALIZADAS: mayúsculas la primera letra y el resto en minúsculas
  - Las **variables**, por último, siempre deben empezar por letra MINÚSCULA.

Nombre	Descripción	Ejemplo
camelCase	Primera palabra, minúsculas. El resto, minúsculas (salvo primera letra). La más usada en JS.	precioProducto
PascalCase	Idem, pero todas las palabras empiezan con la primera letra mayúscula. Se usa en Clases.	PrecioProducto
snake_case	Las palabras se separan con un guión bajo y se escriben siempre en minúsculas.	precio_producto
kebab-case	Las palabras se separan con un guión normal y se escriben siempre en minúsculas.	precio-producto
~~**dot.case**~~	~~Las palabras van en minúsculas separadas por puntos. En JS **no** se puede usar.~~	~~precio.producto~~
Húngara	Prefijo (minúsculas) que indica tipo de dato (n = número, t = texto, ...). Desaconsejada actualmente.	nPrecioProducto

Tipos de datos

- JavaScript es un lenguaje de **tipado débil** porque cuando se declara una variable no se indica su tipo.
- JavaScript es un lenguaje de **tipado dinámico** porque las variables pueden cambiar de tipo en tiempo de ejecución.
- Existen dos tipos principales de datos:
  - **Primitivos (son inmutables):**
    - **Boolean: true o false**
    - **Number: 64 bits en doble precisión**
    - **String: un conjunto de caracteres**
    - **Null: referencia nula**
    - **Undefined: variable declarada pero sin valor asignado**
  - **Objetos (son mutables):**

    Todo lo que no es un tipo primitivo, incluido las funciones.

Tipos de datos. Ejemplos:

```
let numero1=1;          console.log(typeof numero1);      // number
let numero2=1.03;       console.log(typeof numero2);      // number
let boleano=true;       console.log(typeof boleano);      // boolean
let cadena="Hola";      console.log(typeof cadena);       // string
let nulo=null;          console.log(typeof nulo);         // object
                        console.log(nulo);                // null
let indefinido;         console.log(typeof indefinido);   // undefined
                        console.log(indefinido)           // undefined
let objeto={a:1};       console.log(typeof objeto)        // object
```

El operador “typeof” devuelve el tipo de dato de una variable declarada.

```
let a=123;
a="Hola";
console.log(a); // Hola
```

El tipo de la variable puede cambiar en tiempo de ejecución sin problemas.

Tipos de datos. Ejemplos:

En algunos casos, typeof() resulta insuficiente porque en tipos de datos más avanzados simplemente nos indica que son objetos.

Con **constructor.name** podemos obtener el tipo de constructor que se utiliza.

Ejemplos:

```
console.log(text.constructor.name); // String
console.log(number.constructor.name); // Number
console.log(boolean.constructor.name); // Boolean
console.log(notDefined.constructor.name); // ERROR, sólo funciona con variables definidas
```

Declaración de variables

- Se pueden declarar variables de tres formas:
  - Directamente. Ej: dato=123 => permite declarar una variable global en cualquier punto. Es la opción menos recomendada. A partir de ES5 se recomienda utilizar el modo estricto escribiendo en la primera línea del fichero **“use strict”;** con lo que se impide declaraciones directas.
  - Utilizando **var**: permite declarar una variable cuyo ámbito es la función donde se define o global (si se define fuera de una función).
  - Utilizando **let:** es la opción recomendada. La definición de la variable tiene ámbito de bloque de código {…}, de función o global si se define en la raíz del documento. Está disponible a partir de ES6.
- Las constantes se definen con **const** y tienen ámbito de bloque (como con let).

Ámbito de las variables

```
let a=1;        // Ambito global
//...
{               // Ambito de bloque
    let a=2;
    //...
}
function f() {  // Ambito de funcion
    let a=3;
    //...
    if (true)   // Ambito de bloque
    {
        let a=4;
        //...
    }
}
```

Las funciones IIFE (Immediately-invoked function expressions) o **funciones autoinvocadas** permiten encapsular variables declaradas mediante var creando un ámbito local.

```
(function(){
    //...
}());
```

```
(function(n){
    var saludo="Hola "
    console.log(saludo+n); // Hola Juan
}("Juan"));
```

Ámbito de las variables. Hoisting

```
console.log(a);  // undefined
var a=1;
{
    console.log(a);  // 1
}

(function(){
    console.log(a);  // 1
    console.log(b);  // undefined
    var b=2;
    console.log(b)   // 2
}());
console.log(b); // Error: b is not defined
```

```
console.log(a);  // Error: a is not defined
let a=1;
{
    console.log(a);  // 1
}

(function(){
    console.log(a);  // 1
    console.log(b);  // Error: b is not defined
    let b=2;
    console.log(b)   // 2
}());
console.log(b); // Error: b is not defined
```

Analizar el código y ver las diferencias de resultados con “var” a la izquierda y con “let” a la derecha.

https://www.freecodecamp.org/espanol/news/que-es-hoisting-alzar-en-javascript/

Ámbito de las variables. Declaración con “var”.

```
var a = 1;
console.log(1); // 1
{
    console.log(a); // 1
}
(function f() {
    console.log(a) // 1
    if (true) {
        console.log(a) // 1
    }
})();
```

Analizar el código y ver las diferencias de resultados con “var” a la izquierda y a la derecha.

```
var a = 1;
console.log(1); // 1
(function f() {
    var a = 2;
    console.log(a) // 2
    if (true) {
        var a = 3;
        console.log(a); // 3
    }
    console.log(a); // 3
})();
console.log(a); // 1
var a = 5;
console.log(a); // 5
```

Ámbito de las variables. Declaración con “let”.

```
let a = 1;
console.log(1); // 1
{
    console.log(a); // 1
}
(function f() {
    console.log(a) // 1
    if (true) {
        console.log(a); // 1
    }
    console.log(a); // 1
})();
```

```
let a = 1;
console.log(1); // 1
(function f() {
    let a = 2;
    console.log(a) // 2
    if (true) {
        let a = 3;
        console.log(a); // 3
    }
    console.log(a); // 2
})();
console.log(a); // 1
let a = 5; // Error: already declared
```

Analizar el código y ver las diferencias de resultados con “let” a la izquierda y a la derecha.

Operadores

- Operadores aritméticos:
  - Suma: a + b
  - Resta: a - b
  - Multiplicación: a * b
  - División: a / b
  - Resto de división: a % b
  - Exponenciación: a ** b
  - Incremento: ++ (pre y post)
  - Decremento: -- (pre y post)
- Operador ternario:

  (condición)? Instrucción1 : Instrucción 2
- Operador concatenación de string:

  “string1” + “string2”
- Operadores lógicos
  - And: a && b (de corto circuito)
  - Or: a || b (de corto circuito)
  - Not: ! a
- Operadores a nivel de bit:
  - And: a & b
  - Or: a | b
  - Xor: a ^ b
  - Not: ~ a
  - Shift left: a << b
  - Shift right: a << b
- Operadores de comparación :
  - >, <, >=, <=, ==, !==
  - === (igualdad estricta o identidad)
  - !== (desigualdad estricta)

Propiedades globales y funciones globales:

- **Infinity**: representa el valor infinito
- **NaN**: representa un valor no numérico para una variable numérica.
- **null**: representa el valor nulo.

- **isFinite()**: comprueba si es Infinity
- **isNaN()**: comprueba si es NaN
- **parseFloat()**: parsea a float
- **parseInt()**: parsea a entero

Falsy values y Truthy values

```
// falsy values: se evaluan a falso
let a = 0;              console.log(typeof a); console.log(a ? "true" : "false")   // number false
let b = NaN;            console.log(typeof b); console.log(b ? "true" : "false")   // number false
let c = "";             console.log(typeof c); console.log(c ? "true" : "false")   // string false
let d = false;          console.log(typeof d); console.log(d ? "true" : "false")   // boolean false
let e = null;           console.log(typeof e); console.log(e ? "true" : "false")   // undefined false
let f = undefined;      console.log(typeof f); console.log(f ? "true" : "false")   // number false

// truthy values: se evaluan a verdadero
let t = Infinity;       console.log(typeof t); console.log(t ? "true" : "false")   // number true
let u = "0";            console.log(typeof u); console.log(u ? "true" : "false")   // string true
let v = "false";        console.log(typeof v); console.log(v ? "true" : "false")   // string true
let x = function(){};   console.log(typeof x); console.log(x ? "true" : "false")   // function true
let y = [];             console.log(typeof y); console.log(y ? "true" : "false")   // object true
let z = {};             console.log(typeof z); console.log(z ? "true" : "false")   // object true
```

Conversión de tipos:

- En javascript permite cambiar el tipo de las variables dinámicamente (incluso en el modo “use strict”).
- Para convertir entre tipos se puede utilizar los métodos:
  - **parseInt()** y **parseFloat()**: convierten un string a un numero.
  - **toString()**: aplicado a números, devuelve un valor numérico como un string.
  - **eval()**: evalúa un string y ejecuta su contenido.

```
"use strict"
let a=1;
a="hola";
console.log(a);      // hola
let b=parseInt("123.45");
console.log(b);      // 123
let c=parseFloat("123.45");
console.log(c);      // 123.45
let d=46;
console.log(d.toString(2));  // 101110
console.log(d.toString(16)); // 2e
let e=eval("2+4+parseInt('1.2')");
console.log(e);      // 7
```

- Cuando hacemos una asignación o como resultado de una expresión hay que tener en cuenta la “coerción de tipos”

Coerción de tipos

- La coerción es el medio por el cual Javascript trata de convertir al vuelo los valores de cada variable. Ejemplos:

```
let a = "1";
let b = 2;
console.log(typeof a, typeof b); // string number
console.log(a + b); // 12
let c = "2";
let d = 2;
console.log(typeof c, typeof d); // string number
console.log(c - d); // 0
let e ="3";
let f=2;
console.log(typeof e, typeof f); // string number
console.log(e * f); // 6
let g=true;
let h=true;
console.log(typeof g, typeof h); // boolean boolean
console.log(g + h); // 2
```

- Cuando se compara con “==“ se produce coerción de tipos:

```
console.log(true == 1); // true
console.log("" == 0) // true
console.log("0" == 0) // true

console.log(true === 1); // false
console.log("" === 0) // false
console.log("0" === 0) // false
```

- Se recomienda utilizar “===“ en lugar de “==“ porque no hace coerción.
- También se recomienda “!==“ en lugar de “!=“.

Arrays o matrices

```
// Tres formas de crear un array vacío:
let lista1=new Array();
let lista2=Array();
let lista3=[];
// Asigna el primer elemento:
lista1[0]=1;
// Asigna el 6 elemento:
lista1[5]="Pepe";
// Asigna una "propiedad" al array:
lista1["pos1"]="Juan";
lista1.pos2="hola";

console.log(lista1[0]); // 1
console.log(lista1[1]); // undefined
console.log(lista1[5]); // Pepe
console.log(lista1[6]); // undefined
console.log(lista1["pos1"]); // Juan
console.log(lista1.pos1); // Juan
console.log(lista1["pos2"]); // hola
console.log(lista1.pos2); // hola

console.log(lista1.length); // 6
```

- Son colecciones de números, strings, objetos, otros arrays, etc...
- Internamente un array es un objeto y, por ello, se puede utilizar un índice alfanumérico del tipo a[“b”] pero esto crea propiedades de objeto no elementos del array.

```
// Inicializar un array simple:
let tabla1=['a','b','c','d','e','f'];
console.log(tabla1[2]); // c
// Inicializar un array de arrays
let tabla2=[[1,2],[3,4],[5,6]];
console.log(tabla2[2][0]); // 5
// Inicializar un array de tipos difentes
let tabla3=[1,3,4,["A","B"]];
console.log(tabla3[2]); // 4
console.log(tabla3[3][1]); // B
```

Funciones predefinidas típicas de Arrays

- **length**: propiedad que devuelve el número de elementos.
- **concat():** permite concatenar varios arrays.
- **join():** une los elementos de un array y los almacena en un string.
- **pop():** extrae el último elemento de un array.
- **push():** añade un elemento al final del array.
- **shift():** extrae el primer elemento de un array.
- **unshift():** añade un elemento al comienzo del array.
- **reverse():** invierte la posición de los elementos del array.
- **sort():** ordena un array.
- **indexOf():** devuelve la posición de un elemento en el array.
- **lastIndexOf():** devuelve la posición final de un elemento en el array.
- **slice():** devuelve un nuevo array con parte de los elementos del array inicial.
- **splice():** cambia el contenido de un array: inserta, reemplaza o borra elementos del array

Uso de sort() para ordenar arrays.

- Sort ordena (modifica) el array convirtiendo cada elemento en caracteres Unicode. Ejemplo:

```
console.log(['verde', 'azul', 'rojo'].sort()); // ['azul', 'rojo', 'verde']
console.log([20, 1, 2, 10].sort()); // [1,10,2,20] <-- ¡No se ordena correctamente!
console.log([6, -2, 2, -7].sort()); // [-2,-7,2,6] <-- ¡No se ordena correctamente!
```

- Si se desea ordenar un array con números (o cualquier otro objeto) correctamente se debe pasar una función callback como parámetro con el criterio de comparación. Ejemplo:

```
console.log([6, -2, 2, -7].sort(function (a, b) { // [-7, -2, 2, 6] <-- Correcto!
  return a - b;
}));
```

Ver: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/sort
https://www.w3schools.com/js/js_array_sort.asp

Uso de sort() para ordenar arrays.

- Si se desea ordenar un array con números (o cualquier otro objeto) correctamente se debe pasar una función callback como parámetro con el criterio de comparación.

- Ejemplo: Un array con dos elementos "25" y "100" ,la función anónima recibiría a como 25 y b como 100? , entonces resultaría como -75, y como obteniendo un -75, ¿¿la función sabe que 25 es menor que 100 ??

Uso de sort() para ordenar arrays.

- La documentación da unas reglas básicas para este comportamiento.

- Si compareFunction(a, b) es menor que 0, se sitúa a en un índice menor que b. Es decir, a va primero.
- Si compareFunction(a, b) retorna 0, se deja a y b sin cambios entre ellos, pero ordenados con respecto a todos los elementos diferentes.
- Si compareFunction(a, b) es mayor que 0, se sitúa b en un índice menor que a.

Control de flujo

- Las instrucciones se pueden agrupar en bloques entre los caracteres { y }
- El control de flujo se realiza con instrucciones similares a otros lenguajes como Java o C#:
- **if** (expresion) … **else** …
- **switch** (variable) {**case** valor1: … **case** valor2: …}
- **do** {…} **while** (condición)
- **while** (condición) {…}
- **for** (inicio; condición; acción) …
- **for** (indice **in** array/objeto)… → Itera sobre las propiedades/**índices**
- **for** (iterator **of** array/objeto) … → Itera sobre los **valores** (equivale a un foreach)
- **try** …**catch** …

Funciones predefinidas típicas de strings

- **length** : propiedad que devuelve la longitud de una cadena.
- **concat()** o +: concatena dos strings.
- **charAt(pos)** : devuelve el carácter de la posición “pos”.
- **charCodeAt(p):** devuelve el código UNICODE de la posición “pos”.
- **indexOf(sub**) : devuelve la posición de la subcadena.
- **lastIndexOf(sub)** : devuelve la última posición de la subcadena.
- **substr(i,f) :** devuelve la subcadena desde la posición “i” a la posición “f” .
- **toLowerCase() :** convierte la cadena a minúsculas.
- **toUpperCase() :** convierte la cadena a mayúsculas.
- **split(c)** : divide una cadena en diferentes partes separadas por el carácter indicado y las devuelve en un array.

Más funciones predefinidas típicas de strings

- **endsWith()**: determina si un string termina con otro string.
- **startsWith()**: determina si un string empieza con otro string.
- **includes()**: determina si un string contiene otro string.
- **match():** devuelve un array con las ocurrencias.
- **repeat()**: repite un string un numero de veces.
- **replace():** reemplaza un string con otro.
- **trim():** elimina los espacios en blanco.
- **padStart():** añade espacios al inicio.
- **padEnd():** añade espacios al final.

Además hay que tener en cuenta que los strings se comportan como arrays de caracteres →

```
let s="abc";
console.log(s[1]); // b

for (const c of s) { // a
    console.log(c);  // b
}                    // c


for (const k in s) {           // 0 a
    console.log(k+ " "+ s[k]); // 1 b
}                              // 2 c
```

Objeto global Math. Funciones típicas.

- **Math.PI :** propiedad con el númeo Pi: 3.141592…
- **Math.E :** propiedad con el número E: 2.7182818…
- **Math.abs()** : valor absoluto de un número.
- **Math.sin() Math.cos() Math.tan()** : seno, coseno y tangente.
- **Math.exp() Math.log()** : exponenciación y logaritmo.
- **Math.ceil()** : entero >= al argumento.
- **Math.floor()** : entero <= al argumento.
- **Math.round():** redondea a la parte entera más póxima (por arriba o por debajo).
- **Math.pow(b,e):** eleva “b” al exponente “e”.
- **Math.min()** : devuelve el menor de sus argumentos.
- **Math.max()** : devuelve el mayor de sus argumentos.
- **Math.sqrt()** : raíz cuadrada del argumento.
- **Math.random():** devuelve un número aleatorio entre 0 y 1.

Objeto global Date. Funciones típicas.

- **Construtor. Tiene varias sobrecargas:**
  - **new Date()** : crea un objeto con la fecha y hora actual del sistema.
  - **new Date(milisegundos)** : utilizando milisegundos.
  - **new Date(cadenaFech**a): utilizando un string tipo “aaaa-mm-dd”
  - **new Date(año_num,mes_num,dia_num[,hor_num,min_num,seg_num,mils_num])**
- **Funciones:**
  - **now()** : Devuelve el valor numérico correspondiente a la fecha y hora actual.
  - **parse() :** Parsea una string tipo “aaaa-mm-dd” y devuelve una fecha.
  - **getFullYear()** : Devuelve el año de la fecha.
  - **getMonth()** : Devuelve el mes de la fecha.
  - **getDate()** : Devuelve el día del mes de la fecha.
  - **getDay()** : Devuelve el día de la semana (0=domingo … 6:sábado).
  - **getHours(), getMinutes(), getSeconds(), getMilliseconds()** : devuelven hora, min, …
  - **getTime()** : Devuelve el valor numérico (timespam) en ms correspondiente a la fecha.
  - Además existen varias funciones “set” para modificar las fechas.

Template strings

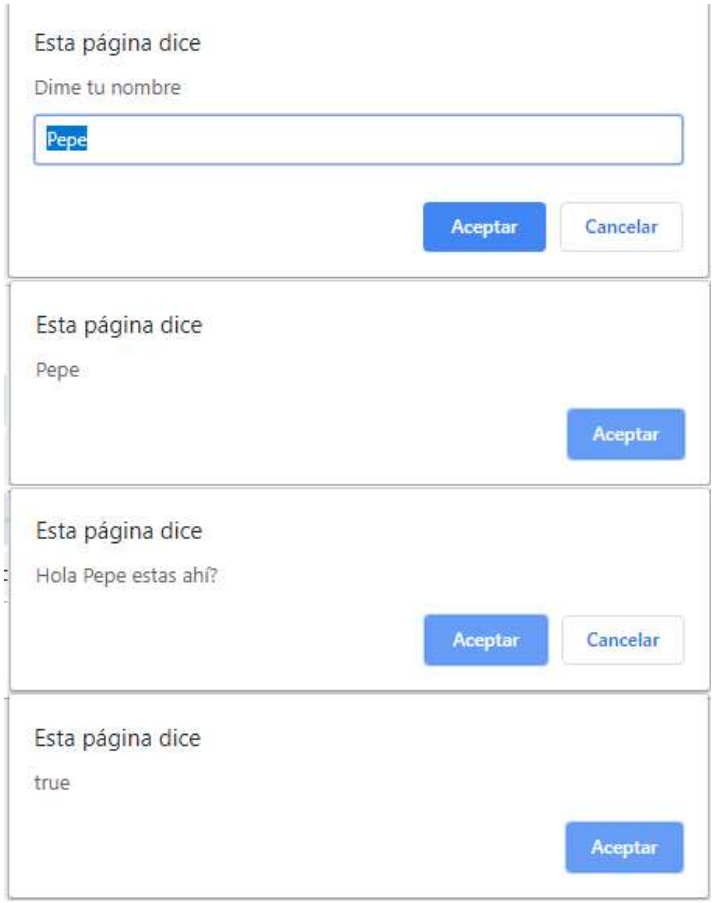
- Los “template strings” introducidos en ES6 permiten funcionalidad adicional a los strings clásicos, por ejemplo, interpolación de strings, escribir una cadena en varias líneas o “Tagged Templates” (plantillas etiquetadas):

```
// Interpolación:
const a = 1, b = 2;
console.log(`Suma: ${a + b}`);  // Suma: 3

// String de varias lineas:
const s=`linea1
linea2`;
console.log(s)      // linea1
                    // linea2

// Definición de función para uso de plantillas etiquetadas:
function foo(texto,p1,p2,p3){
    console.log(texto,p1,p2,p3); // [ 'La suma de ', ' y ', ' es ', '' ] 1 2 3
    return `La suma es: ${p1+p2}`;
}
let res=foo`La suma de ${a} y ${b} es ${a+b}`;  // Uso de la plantilla etiquetada:
console.log(res); // La suma es 3
```

Funciones alert, prompt y confirm

- Permiten interactuar con el usuario. Solo se permiten en un script para navegador:
  - Muestra un mensaje: **alert(message);**
  - Solicita un dato: **result = prompt(title, [default]);**
  - Solicita un Si o No: **result = confirm(question);**

```
<script>
    let nombre=prompt("Dime tu nombre","Pepe");
    alert(nombre);
    alert(confirm(`Hola ${nombre} estas ahí?`));
</script>
```



Ejercicio 0

- Probar todas las funciones vistas anteriormente en cuatro ficheros js.
  - Funciones de Arrays
  - Funciones de Strings
  - Funciones matemáticas
  - Funciones de Date

Ejercicio 1 (ej1.html)

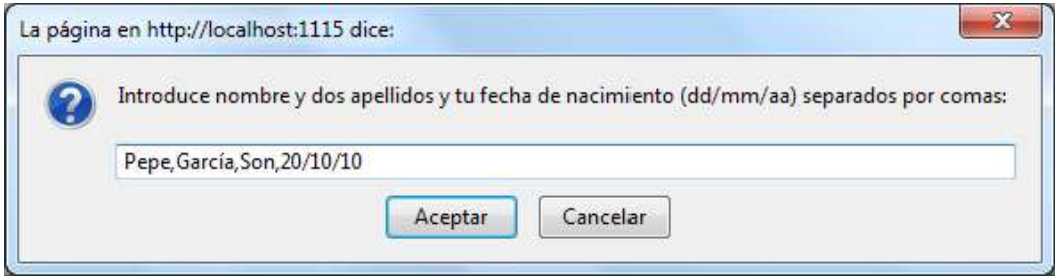
- Crear una página web que contenga un script que permita:
  1. Introducir mediante “prompt” el número de filas y el número de columnas de una tabla.
  2. Crear las celdas de una tabla generada mediante javascript (*) en las que aparezcan sus índices con el siguiente formato:

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,10	0,11	0,12	0,13	0,14	0,15	0,16	0,17	0,18	0,19	0,20	0,21	0,22
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	1,11	1,12	1,13	1,14	1,15	1,16	1,17	1,18	1,19	1,20	1,21	1,22
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10	2,11	2,12	2,13	2,14	2,15	2,16	2,17	2,18	2,19	2,20	2,21	2,22
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10	3,11	3,12	3,13	3,14	3,15	3,16	3,17	3,18	3,19	3,20	3,21	3,22
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10	4,11	4,12	4,13	4,14	4,15	4,16	4,17	4,18	4,19	4,20	4,21	4,22
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9	5,10	5,11	5,12	5,13	5,14	5,15	5,16	5,17	5,18	5,19	5,20	5,21	5,22

(*) Para generar las celdas se puede utilizar la función document.write() que permite insertar código html en la propia página.

Ejercicio 2 (ej2.html)

- Crear una página web que contenga un script que permita:
  1. Introducir mediante prompt el nombre, dos apellidos y la fecha de nacimiento (separados por comas) con el siguiente formato:

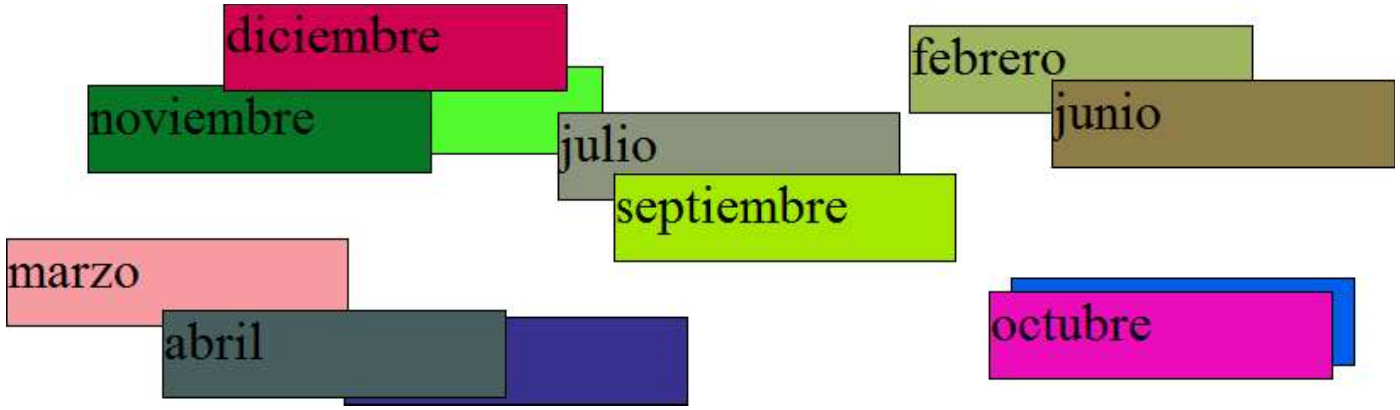


  2. Separar los datos introducidos y validar la fecha para generar el html que visualice el resultado siguiente (si la fecha no es correcta, se visualizará un mensaje de error apropiado):

Nombre	Pepe
Primer apellido	Garcia
Segundo apellido	Son
Fecha	miércoles, 20 de octubre de 2010 0:00:00

Ejercicio 3 (ej3.html)

- Crear el código javascript que permita:
  1. Crear una tabla con los nombres de los meses del año y doce capas ( ) (una por cada mes) en las que aparezca el nombre del mes.
  2. Asignar a cada capa un estilo para posicionarla aleatoriamente en la pantalla dentro de un rango de 800 pixels de ancho y 200 de alto.
  3. Asignar a cada capa un color de fondo aleatorio mediante la regla de estilo: **background-color:rgb(red,green,blue)(*).**



(*)red, green y blue son número comprendidos entre 0 y 255.

Expresiones regulares

- Las expresiones regulares (o regex) permiten trabajar con cadenas de texto de forma eficaz de tres formas diferentes:
  - Buscar texto en un string
  - Reemplazar un string con otros
  - Extraer un string de otro
- En javascript se pueden definir las expresiones regulares de dos formas diferentes:
  - Mediante un objeto.
  - Mediante un literal.

En el ejemplo re1 y re2 son dos expresiones regulares equivalentes. El patrón de búsqueda es “hola”.
El método test devuelve un bool indicando si el patrón aparece en el string.

```
const re1 = RegExp('hola');
const re2 = /hola/;
console.log(re1.test('El mensaje es "hola mundo"')); // true
console.log(re2.test('El mensaje es "hello mundo"')); // false
```

Expresiones regulares

- Para buscar en un string que comienza con un patrón se utiliza “^”.
- Para buscar en un string que termina en un patrón se utiliza “$”.
- “.” representa un carácter cualquiera que no sea nueva línea “\n”.
- “.*” es un patrón para cero o más caracteres.
- “[a-z]” representa un carácter cualquiera dentro del rango de la “a” a la “z”.
- Otros rangos posibles: “[xyz]”, “[0-9]”, “[A-Z]”, “[o-z]” o “[A-Za-z]

```
console.log(/^hello/.test('hola mundo')); // false
console.log(/world$/.test('hola mundo')); // false
console.log(/^h.*o$/.test('hola mundo')); // true
console.log(/^[0-9]/.test('hola mundo')); // false
console.log(/[u-v]/.test('hola mundo')); // true
```

El símbolo “^” dentro de un rango niega el rango. Ejemplo: [^0-9] representa un carácter no numérico.

Expresiones regulares

- “\d” coincide con un dígito, equivalente a [0-9]
- “\D” coincide con un carácter que no sea un dígito, equivalente a [^0-9]
- “\w” coincide con un carácter alfanumérico (más guion bajo), equivalente a [A-Za-z_0-9]
- “\W” coincide con un carácter no alfanumérico, cualquier cosa excepto [^A-Za-z_0-9]
- “\s” coincide con un carácter de espacio en blanco: espacios, tabulaciones, nuevas líneas y espacios Unicode
- “\S” coincide con un carácter que no sea un espacio en blanco
- “\0” coincide con nulo
- “\n” coincide con un carácter de nueva línea
- “\t” coincide con un carácter de tabulación
- “\uXXXX” coincide con un carácter Unicode con el código XXXX
- “.” coincide con un carácter que no sea un carácter de nueva.

Expresiones regulares

- “|” representa “or” para buscar por varios patrones posibles. Ejemplo:

```
console.log(/^hello|^hola/.test('hola mundo')); // true
console.log(/^[8-9]|o$/.test('hola mundo')); // true
```

- Cuantificadores:
  - “?” representa 0 o 1 elementos.
  - “+” representa 1 o más elementos.
  - “*” representa 0 o más elementos.
  - “{n}” representa n elementos.
  - {n,m} representa entre n y m elementos
  - {n,} representa “al menos” n elementos

```
console.log(/^a\d?/.test('abc')); // true
console.log(/^a\d?/.test('ab3')); // true
console.log(/^a\d/.test('abc')); // false
console.log(/^a\d/.test('ab3')); // false
console.log(/\d{3}/.test('h34a')); // false
console.log(/\d{3}/.test('h346a')); // true
console.log(/[a-c]{3,}/.test('--cab--')); // true
```

Expresiones regulares

- Los paréntesis (…) sirven para agrupar parte un patrón y opcionalmente indicar la repetición de dicho grupo. Ejemplo:

```
console.log(/(abc){2}(.\d)/.test('--abcabcx4--')); // true
```

- El primer grupo “(abc)” se tiene que repetir dos veces.
- El segundo grupo “(.\d)” debe aparecer una sola vez (seguido del primer grupo)
- Los grupos también sirven para capturar las coincidencias. Esto se hace con las dos funciones equivalentes: **String.match(RegExp)** y **RegExp.exec(string)**

```
const result = '--abcabcx4--'.match(/(abc){2}(.\d)/);
console.log(result.length); // 3
console.log(result[0]); // abcabcx4 --> texto total coincidente
console.log(result[1]); // abc --> captura del primer grupo
console.log(result[2]); // xa --> captura del segundo grupo
```

Expresiones regulares

- Para reemplazar parte de un string por otro se puede utilizar el método replace() de un string. Solo reemplaza la primera ocurrencia. Ejemplo:

```
const r = 'My dog is a good dog';
console.log(r.replace('dog', 'cat')); // My cat is a good dog
```

- Con una expresión regular podría ser (la “g” indica búsqueda global):

```
const r = 'My dog is a good dog';
console.log(r.replace(/dog/g, 'cat')); // My cat is a good cat
```

- Mas ejemplos:

```
const r1 = 'My dog is a good. My cat is good';
console.log(r1.replace(/(dog|cat)/g, 'pet')); // My pet is good. My pet is good
const r2 = 'AAAsssAAssxAkkkkAAAAAA';
console.log(r2.replace(/A{2,}/g, 'B')); // BsssBssxAkkkkB
```