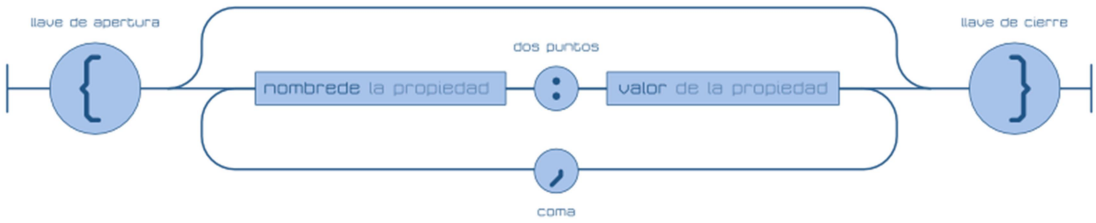
Manipulación del Formato JSON en JavaScript

JSON es un formato de intercambio de datos basado en texto, es decir, es fácil de leer para una persona además de para un programa. Su nombre corresponde con las siglas en inglés de JavaScript Object Notationy. Aunque es un formato muy cercano en sintaxis a JavaScript, por ser muy sencillo de tratar para codificar datos de objetos o para obtenerlos, se utiliza también en muchos otros lenguajes de programación (C, C++, Java, Python…) como una alternativa, por ejemplo, a XML, que tiene un objetivo similar pero que, por incluir más metainformación, necesita más texto y por tanto ocupa más, consume más ancho de banda y necesita más recursos para codificar y decodificar la información que JSON.

Los datos de un objeto codificados en JSON están encerrados **entre llaves, las distintas propiedades que se incluyen dentro se separan por comas y los nombres de las propiedades preceden al valor**, del que **se separan por dos puntos**.



Los nombres de las propiedades deben **entrecomillarse usando la doble comilla** y pese a que se admiten algunos caracteres especiales para los nombres (como las tildes) no es recomendable para evitar el criterio de algunos motores de análisis y para evitar conflictos al usar la propiedad especialmente con la sitaxis de punto (objeto.propiedad)

```
{
        "dispositivo":"EA4-CD-145",
        "tension":12.1,
        "intensidad":0.99,
        "temperatura":19.25,
        "deteccion":false,
        "tiempo_uso":158.36
}
```

En el ejemplo de arriba puede verse un objeto que tiene seis propiedades con valores sencillos, numéricos, booleanos o de texto; pero en el formato JSON, las propiedades pueden adquirir también como valor un objeto (incluyendo el «objeto especial» null) o una matriz además de una

cadena de texto (un texto entre comillas), un número (en diferentes formatos) o un valor booleano (true o false).

Al igual que ocurre con el lenguaje JavaScript (JSON es un subconjunto de JavaScript) para expresar una matriz se encierran entre corchetes sus valores y se separan por comas. El caso más sencillo se ilustra en el ejemplo de abajo; se trata de una matriz unidimensional, un vector, compuesto por valores numéricos.

```
{
	"temperatura":[10.00,12.50,15.75,18.25,19.50,20.00,21.75,22.00,22.00,22.25]
}
```

En JavaScript no es necesario que todos los elementos de una matriz sean del mismo tipo, se pueden mezclar, por ejemplo, cadenas de texto y valores numéricos. Un vector, una matriz unidimensional, puede ser también uno de los elementos de otra matriz, lo que permite construir matrices multidimensionales de longitudes variables. El siguiente ejemplo muestra un objeto con tres propiedades: la primera una matriz bidimensional de longitudes fijas, longitudes variables en la segunda y la tercera formada por una matriz con valores de tipos diferentes.

```
{
	"coordenada":
	[
		[78.59,12.33],
		[85.23,8.01],
		[90.95,6.12],
		[92.50,3.86],
		[96.12,-1.55],
		[98.60,-3.14]
	],
	"impacto":
	[
		[5,20,10,36,10,2],
		[33,41,43],
		[44,44,44,49,51,52,60,62,59],
		[12,11,15,16]
	],
	"entrada":
	[
		"T26-A",
		[63.15,57.37],
		14420.122256,
		true
	],

}
```

En el siguiente ejemplo se usan otros objetos como valores de las propiedades del objeto principal. No hay un límite en el nivel de anidamiento así que, a su vez, los objetos que sean valores de propiedades del principal

también pueden tener otros objetos como valores de sus propiedades y así sucesivamente.

```
{
	"pir":
	{
		umbral:0.15,
		ultima:258982.67,
		deteccion:false
	},
	"termometro":
	{
		actual:19.25,
		minima:8.05,
		maxima,22.60
	},
	"vibracion":false,
	"distancias":
	[
		12.60,
		9.10,
		12.60,
		8.90
	]
}
```

Igual que ocurre con JavaScript, para incluir determinados caracteres dentro de una cadena de texto se utilizan los códigos de escape. La forma más genérica consiste en usar códigos unicode expresados con la barra de escape, la letra u y cuatro dígitos hexadecimales con el formato "\u263A".

Los caracteres estándar, presentes en la mayoría de los lenguajes (similares a C) son los siguientes

- Retroceso \b Código ASCII 8 (0x08)
- Tabulador \t Código ASCII 9 (0x09) Se suele representar como HT (horizontal tabulator)
- Nueva línea \n Código ASCII 10 (0x0A) Se suele representar como LF (line feed)
- Nueva página \f Código ASCII 12 (0x0C) Se suele representar como FF (form feed)
- Retorno \r Código ASCII 13 (0x0D) Se suele representar como CR (carriage return)
- Comillas \" Código ASCII 34 (0x22)
- Barra de división \/ Código ASCII 57 (0x2F)
- Barra invertida \\ Código ASCII 134 (0x5C)
- Código unicode \uXXXX

Derivado de C (y relacionado con Unix y similares, como GNU/Linux) el fin de línea suele representarse con \n y es el que debe elegirse en JSON pero es

interesante recordar que algunos sistemas operativos prefieren otras alternativas. Windows suele representar con \r\n las terminaciones de línea y Mac OS (antes de que existiera OS X) solía utilizar \r

Con respecto al formato numérico, la referencia también es JavaScript. El separador decimal es el punto, el guión (ASCII 45 0x2D) se usa como signo negativo y la notación en forma exponencial (notación científica) utiliza E como indicador (que puede ser mayúscula o minúscula)

```
{
        "sondas":["ZA-01","ZB-01","ZB-02","ZB-03"],
        "maximos":[5.612E+6,1.3774E+8,1.034E+9,2.992E+7],
        "minimos":[1.25E-3,1.01E-3,1.2E-7,2.351E-3],
        "ayuda":"Antes de colocar\nel sensor en el medio\nAsegúrese de verificar\nEl calibrado previo"
}
```

Para usar los datos, que se habrán obtenido como un texto, los diferentes lenguajes de programación disponen de funciones de análisis y asignación.

A partir del contenido del archivo json podemos generar un objeto JavaScript. Una vez generemos el objeto JavaScript ya podremos manipularlo de forma cómoda con notación de punto y notación de array.

JSON.parse utiliza un objeto predefinido JavaScript (el objeto JSON) para convertir la cadena de texto con la definición del objeto en un objeto en sí.

```
var objeto_json = JSON.parse(jsonResponse);
```

Una vez tenemos el objeto ya podemos acceder a sus propiedades con notación de punto y/o notación de array. Por ejemplo, podríamos obtener el nombre del primer país de la lista de países escribiendo:

```
var nombrePrimerPais = objeto_json.listadoPaises.pais[0].nombre;
```

La propiedad listadoPaises consta de una serie de elementos a los que podemos acceder por índice. Para acceder al primer elemento usaríamos el índice cero. Cada elemento país está compuesto por nombre, capital, textoCapital y un array de ciudades importantes denominado ciudadImportante. Para acceder al nombre del primer país (España) escribimos objeto_json.listadoPaises.pais[0].nombre. Si quisiéramos acceder a la tercera ciudad del primer país escribiríamos:

```
objeto_json.listadoPaises.pais[0].ciudadImportante[2]
```

Una vez tenemos el objeto con la definición del archivo json extraemos otro objeto asemejable a un array que contiene la información de los países.

```
paisesRecibidos = objeto_json.listadoPaises.pais;
```

Este nuevo objeto lo recorremos con un bucle for extrayendo la propiedad nombre de cada elemento recorrido:

```
var nombrePais = objeto_json.listadoPaises.pais[i].nombre;
```

Cuando comprobamos que el nombre del elemento coincide con el nombre buscado, extraemos el valor de ciudadImportante, que es un array de elementos, mediante la instrucción:

```
var ciudadesPais = objeto_json.listadoPaises.pais[i].ciudadImportante;
```

Una vez tenemos el array ciudadesPais simplemente tenemos que recorrerlo para mostrar la información por pantalla.

Si quisiéramos convertir un objeto JSON en texto utilizaríamos la función **stringify** de la siguiente forma:

```
var texto=JSON.stringify(estado);
```

El formato JSON es bastante flexible pero, principalmente por estar basado en texto para ser humanamente legible, presenta algunos inconvenientes. El primero, es que necesita más memoria y consume más ancho de banda de lo imprescindible.

El segundo inconveniente se plantea al tratar con datos netamente binarios (por ejemplo imágenes) Cuando se trate de porciones pequeñas se puede resolver con lo explicado para resolver el formateo de caracteres especiales pero si se trata de codificar información de ciertas dimensiones será necesario embeberla dentro del formato JSON utilizando otro formato de texto. La codificación más usada para resolver este aspecto es **Base64** ya que muchos lenguajes disponen de librerías para convertir la información en una y otra dirección.

EVITAR EL CACHEADO DE RESPUESTAS

Los datos recuperados usando **GET** pueden quedar cacheados en los navegadores. Esto significa que, si repetimos una petición para recuperar datos con GET que ha sido realizada anteriormente, podemos obtener los mismos resultados en el navegador incluso si los datos han cambiado en el servidor (debido al cacheado en el navegador).

Para evitar el cacheado de respuesta hemos usado esta expresión:

```
objetoXHR.open("GET","listadoPaises.json?nocache=” + (new Date()).getTime());
```

Aquí la url invocada es una url a la que se añade una terminación aleatoria irrelevante para el resultado pero que da lugar a que el navegador interprete la petición como si se tratara de una nueva petición evitando el cacheado.

Si se realizan las peticiones con POST no ocurre este problema.