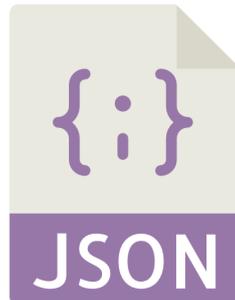




# Análisis básico de datos en formato **JSON**



## Introducción

1.-Bloque para cargar e interpretar datos de texto en formato JSON

2.- Acceso a los datos

2.1.- Listado de datos clave/valor (diccionario):

2.2.- Listado de datos en forma de vector (array):

3.-Comprobar la existencia de claves

4.-Recorrer los valores clave/valor

5.-Ejemplos prácticos

5.1.- Archivo de configuración en SD

5.1.-Intercambio de datos entre placas ESP32 STEAMakers mediante MQTT

5.3.-Analizar respuesta desde API (obtenida con cliente HTTP)

## ANEXO

Juanjo López

## Introducción

JSON (JavaScript Object Notation) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o vice versa)

Un texto en formato JSON es una cadena cuyo formato recuerda al de los objetos literales JavaScript. Es posible incluir los mismos tipos de datos básicos dentro de un JSON que en un objeto estándar de JavaScript - cadenas, números, arreglos, booleanos, y otros literales de objeto. Esto permite construir una jerarquía de datos, como ésta:

Referencia:

<https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON>

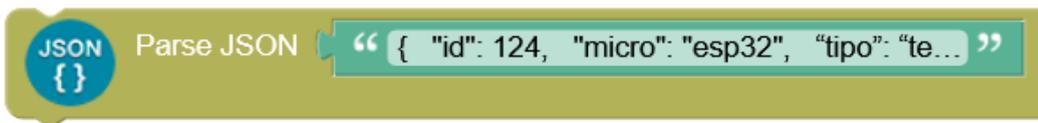
Ejemplo de datos en formato JSON:

```
{
  "id": 124,
  "micro": "esp32",
  "tipo": "temperatura",
  "localizacion": "comedor",
  "valor": 22.5,
  "permisos": {
    "lectura": true,
    "escritura": false
  }
  "autorizados": ["maría", "juan", "pedro", "laura"],
  "historico": [22.5, 21.3, 23.4, 19.5],
  "activo": true
}
```

Podemos crear una estructura en formato JSON de ilimitados niveles, incluyendo dentro de cada elemento nuevas ramificaciones.

En ArduinoBlocks se ha buscado implementar los bloques de la forma más sencilla de uso, lo cual no optimiza el uso interno de la memoria y la librería en pro de la sencillez y facilidad de uso. Se plantea el uso en casos no muy complejos, pues todo es puramente con fines didácticos.

## 1.-Bloque para cargar e interpretar datos de texto en formato JSON:



Internamente analiza el texto en formato JSON y crea una estructura para que podemos acceder fácilmente a los datos.

## 2.- Acceso a los datos

Básicamente tenemos dos tipos de estructuras:

### 2.1.- Listado de datos clave/valor (diccionario):

```
{ "clave1": "valor1", "clave2": "valor2", "clave3": "valor3", ... }
```

Para acceder a un valor accederemos por el nombre de la clave correspondiente. En el ejemplo si queremos obtener el campo "id" como valor numérico:



Podemos guardarlo directamente en una variable numérica o trabajar con el valor como un número.



De la misma forma podemos acceder a datos de tipo texto o booleanos:



Como caso especial cuando obtengamos el valor de un campo a través de su clave y ese campo tengo un sub-objeto JSON en su valor, obtendremos en forma de texto toda la cadena JSON de su interior. Normalmente para volver a procesar (parsear)

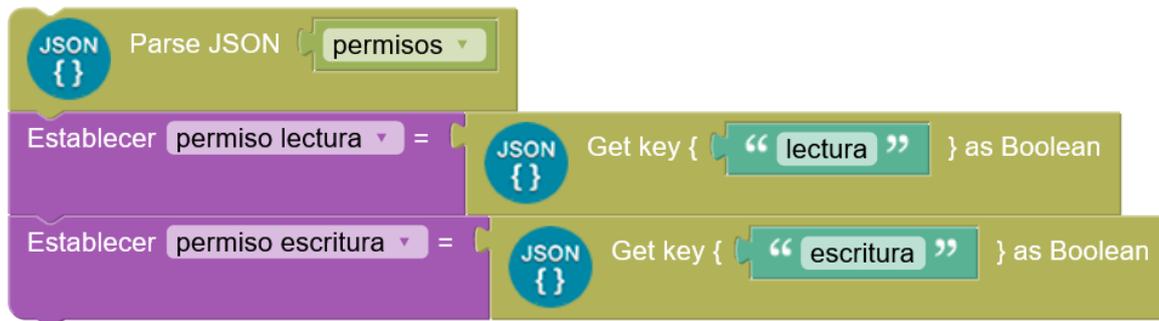
En el ejemplo:



La variable permisos contendrá el valor:

```
{  
  "lectura":true,  
  "escritura":false  
}
```

Y podemos volver a procesar para obtener fácilmente los valores:



De esta forma podemos ir analizando los distintos niveles internos de la estructura JSON del documento.

**ATENCIÓN:** En el momento que volvemos a “parsear” un nuevo texto en formato JSON perdemos la información anterior, por lo que debemos recuperar los valores que nos interese del nivel de análisis correspondiente e ir recorriendo el interior de la estructura reanalizando los subniveles para acceder a sus valores.

## 2.2.- Listado de datos en forma de vector (array):

[ valor1, valor2, valor3, ...]

Para acceder a un valor accederemos por el índice de posición dentro del vector (numérico)

En el ejemplo:



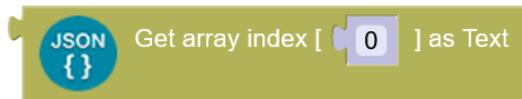
Establecer usuarios autorizados = JSON Get key { " autorizados " } as Text

Obtenemos el valor:

["maría","juan","pedro","laura"]

Y podemos acceder a cada valor por su posición:

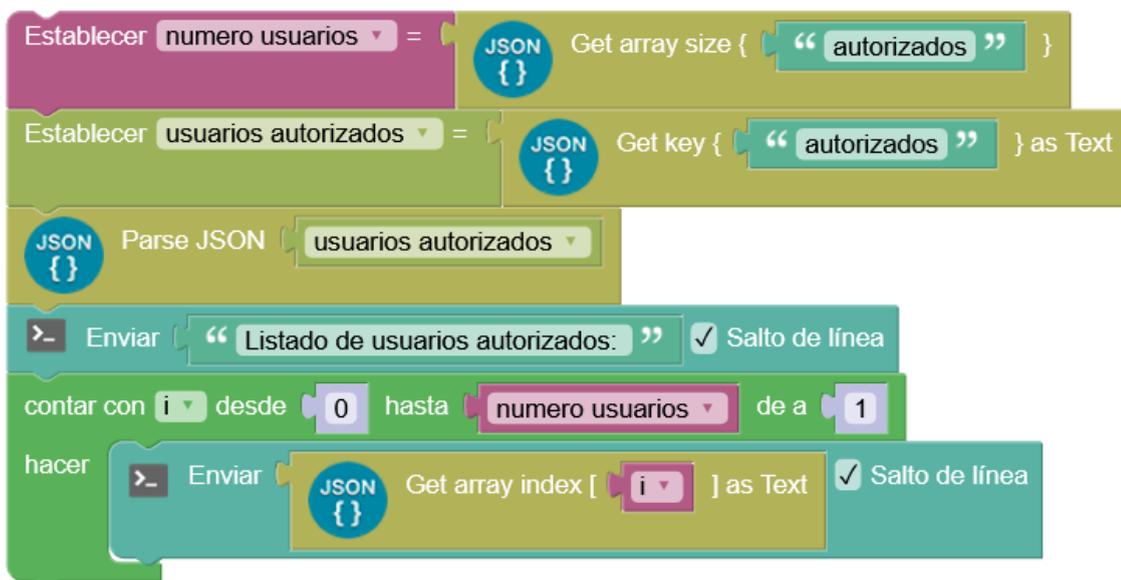
- 0 -> "maría"
- 1 -> "juan"
- 2 -> "pedro"
- 3 -> "laura"



JSON Get array index [ 0 ] as Text

En la mayoría de casos el número de elementos en el vector será variable por lo que podemos obtener su tamaño y recorrer los valores con un sencillo bucle.

En el ejemplo anterior:



Establecer numero usuarios = JSON Get array size { " autorizados " }

Establecer usuarios autorizados = JSON Get key { " autorizados " } as Text

JSON Parse JSON usuarios autorizados

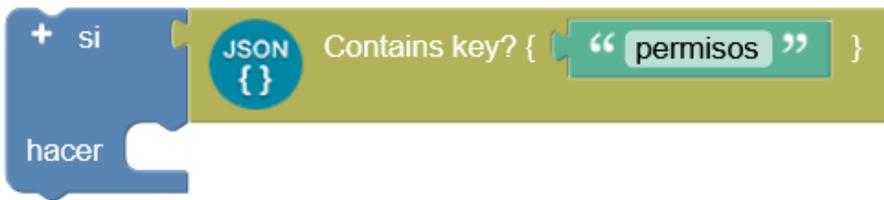
>\_ Enviar " Listado de usuarios autorizados: " Salto de línea

contar con i desde 0 hasta numero usuarios de a 1

hacer >\_ Enviar JSON Get array index [ i ] as Text Salto de línea

### 3.-Comprobar la existencia de claves

En algunos casos debemos comprobar si una determinada clave se encuentra en el documento JSON o no, para ello podemos usar este bloque:



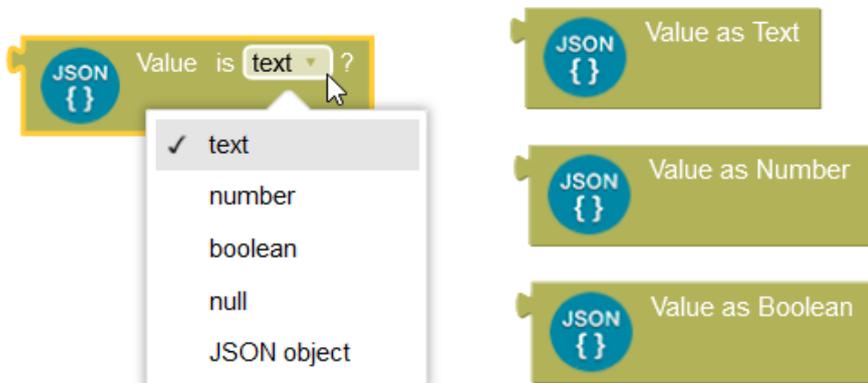
### 4.-Recorrer los valores clave/valor

En algunos casos podemos necesitar recorrer los pares clave/valor del objeto JSON sin saber a priori sus nombres.

Bloque para recorrer la lista clave/valor, y el bloque para obtener el nombre de la clave en cada iteración (el bloque de obtener la clave sólo tendrá un valor válido si lo usamos dentro del bucle de iteración):



En este caso necesitaremos averiguar el tipo de dato y poder acceder al valor según el tipo:



Ejemplo: recorre cada clave/valor y lo muestra por la consola según el tipo de dato

The image shows a Scratch script designed to iterate through a JSON object and display its key-value pairs in a structured format. The script is contained within a 'For each key:value' loop block.

- Loop Start:** A 'For each key:value' block with a 'JSON {}' icon.
- Iteration:** A 'hacer' loop block that repeats the following steps for each key-value pair:
  - Key:** An 'Enviar' block with a 'JSON {}' icon and the text 'Key', followed by a 'Salto de línea' checkbox.
  - Separator:** An 'Enviar' block with the text '->' and a 'Salto de línea' checkbox.
  - Text Check:** A 'si' block with a 'JSON {}' icon and the text 'Value is text ?'.
    - True:** An 'Enviar' block with the text 't:' and a 'Salto de línea' checkbox, followed by another 'Enviar' block with a 'JSON {}' icon and the text 'Value as Text', with a checked 'Salto de línea' checkbox.
  - Number Check:** A 'sino si' block with a 'JSON {}' icon and the text 'Value is number ?'.
    - True:** An 'Enviar' block with the text 'n:' and a 'Salto de línea' checkbox, followed by another 'Enviar' block with a 'JSON {}' icon and the text 'Value as Number', with a checked 'Salto de línea' checkbox.
  - Boolean Check:** A 'sino si' block with a 'JSON {}' icon and the text 'Value is boolean ?'.
    - True:** An 'Enviar' block with the text 'b:' and a 'Salto de línea' checkbox, followed by another 'Enviar' block with a 'JSON {}' icon and the text 'Value as Boolean', with a checked 'Salto de línea' checkbox.
  - JSON Object Check:** A 'sino si' block with a 'JSON {}' icon and the text 'Value is JSON object ?'.
    - True:** An 'Enviar' block with the text 'json:' and a 'Salto de línea' checkbox, followed by another 'Enviar' block with a 'JSON {}' icon and the text 'Value as Text', with a checked 'Salto de línea' checkbox.
  - Default:** A 'sino' block with an 'Enviar' block containing the text '<desconocido>' and a checked 'Salto de línea' checkbox.
- Loop End:** A final 'Enviar' block with a 'JSON {}' icon and a 'Salto de línea' checkbox.

## 5.-Ejemplos prácticos

### 5.1.- Archivo de configuración en SD

En la tarjeta microSD insertada en la ranura de la placa ESP32 STEAMakers tendremos un archivo llamado "config.json" con datos en formato JSON de parámetros para nuestra aplicación

```
{  
  "mensajeinicial": "App de control 1.0",  
  "indicador": true,  
  "servo": {  
    "activo":true,  
    "posicion":200  
  }  
}
```

The Scratch code is organized into an 'Inicializar' (Initialize) script. It begins with an 'OLED' block to initialize I2C at address 0x3C and display the text 'Mostrar automáticamente'. This is followed by an 'SD Iniciar' block. The core logic involves reading the 'config.json' file from the SD card using the 'Leer contenido como texto' block. The JSON data is then processed through a 'Parse JSON' block. The resulting values are used to set variables: 'mensajes inicial' (text), 'indicador led' (boolean), 'aux' (text), 'motor activo' (boolean), and 'motor posicion' (number). After parsing, the 'aux' variable is also processed. The code then clears the OLED display. A 'Texto X Y' block displays the 'mensajes inicial' on the OLED. A conditional 'si' block checks the 'indicador led' variable; if true, the LED on pin 25 (D3) is turned ON, otherwise it is turned OFF. Another conditional 'si' block checks the 'motor activo' variable; if true, a servo motor on pin 17 (D4) is moved to the position specified in 'motor posicion' with a 0ms delay.

## 5.1.-Intercambio de datos entre placas ESP32 STEAMakers mediante MQTT

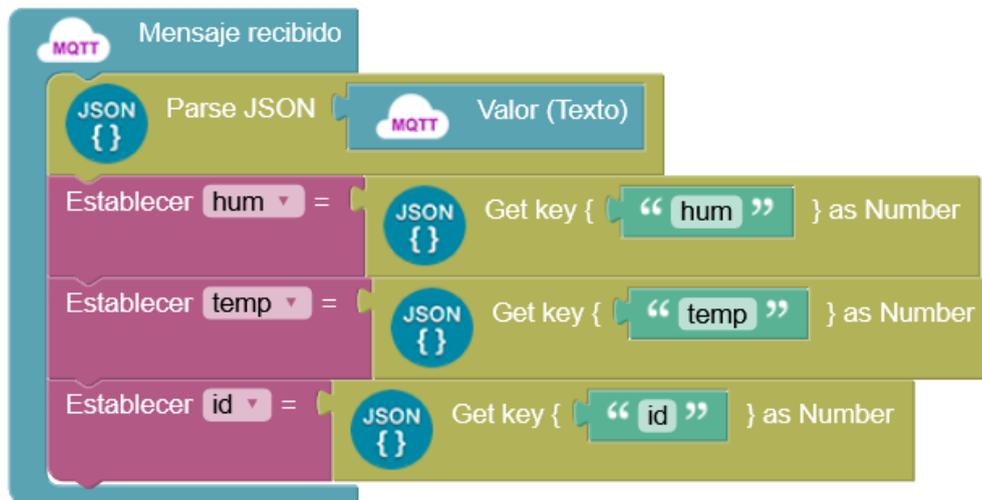
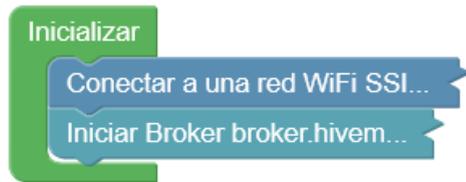
Formato de los datos en JSON:

```
{  
  "temp": xx,  
  "hum": yy,  
  "id": id  
}
```

a) Placa emisora:

```
Iniciar  
  Conectar a una red WiFi SSI...  
  Iniciar Broker broker.hivem...  
  
Bucle  
  Ejecutar cada 5000 ms  
    Establecer temp = DHT-11 Temperatura °C Pin 26 (D2)  
    Establecer hum = DHT-11 Humedad % Pin 26 (D2)  
    Establecer json data = crear texto con  
      {  
      "temp":  
      temp  
      , "hum":  
      hum  
      , "id": 48  
      }  
    MQTT Publicar Tema "ab/json/data" Valor json data
```

b) Placa receptora:



### 5.3.-Analizar respuesta desde API (obtenida con cliente HTTP)

a) Obtener un chiste aleatorio desde la API:

[https://official-joke-api.appspot.com/random\\_joke](https://official-joke-api.appspot.com/random_joke)

Formato JSON, ejemplo:

```
{  
  "id":335,  
  "type":"general",  
  "setup":"Why did the cookie cry?",  
  "punchline":"Because his mother was a wafer so long"  
}
```

**Inicializar**

- Conectar a una red WiFi SSI...
- Iniciar Baudios 115200
- Enviar " Un chiste cada 30s... ;)"  Salto de línea
- Enviar " "  Salto de línea

**Bucle**

Ejecutar cada 30000 ms

**JSON** Parse JSON

**http://** Petición GET

URL " https://official-joke-api.appspot.com/random\_joke "

BasicAuth: Usuario " " "

Clave " " "

- Enviar " NEWJOKE! Type: "  Salto de línea
- Enviar **JSON** Get key { " type " } as Text  Salto de línea
- Enviar " "  Salto de línea
- Enviar **JSON** Get key { " setup " } as Text  Salto de línea
- Enviar " .... "  Salto de línea
- Esperar 5000 milisegundos
- Enviar **JSON** Get key { " punchline " } as Text  Salto de línea
- Enviar " :) :) :) :) "  Salto de línea

Resultado en la consola:

```

Un chiste cada 30s... ;)

NEW JOKE! Type: general

What do you give to a lemon in need?
....
Lemonaid.
:) :) :) :)
NEW JOKE! Type: general

What kind of music do planets listen to?
....
Nep-tunes.
:) :) :) :)

```

b) Obtener datos meteorológicos desde OpenWeather

En esta web debemos registrarnos y obtener una API KEY

<https://openweathermap.org/api>

También es interesante obtener las coordenadas GPS en formato decimal del lugar de donde queremos consultar los datos meteorológicos

<https://www.latlong.net/convert-address-to-lat-long.html>

Ejemplo: *lat: 38.626369 , long: -0.570284*

Un ejemplo de petición para obtener datos básicos:

<https://api.openweathermap.org/data/2.5/weather?lat=38.626369&lon=-0.570284&units=metric&appid=APIKEY>

Ejemplo de datos obtenidos:

```
{ "coord": { "lon": -0.5703, "lat": 38.6264 }, "weather": [ { "id": 803, "main": "Clouds", "description": "broken clouds", "icon": "04n" } ], "base": "stations", "main": { "temp": 286.35, "feels_like": 285.27, "temp_min": 285.38, "temp_max": 286.74, "pressure": 1023, "humidity": 59, "sea_level": 1023, "grnd_level": 935 }, "visibility": 10000, "wind": { "speed": 4.03, "deg": 306, "gust": 10 }, "clouds": { "all": 54 }, "dt": 1668974142, "sys": { "type": 2, "id": 2011177, "country": "ES", "sunrise": 1668926984, "sunset": 1668962779 }, "timezone": 3600, "id": 2516480, "name": "Ibi", "cod": 200 }
```

The image shows a Scratch script for fetching weather data. It starts with an 'Iniciar' block containing 'Conectar a una red WiFi SSL...', 'Iniciar Baudios 115200', and 'Enviar "OpenWeather wind info:" with a line break. A 'Bucle' block follows, set to 'Ejecutar cada 30000 ms'. Inside the loop, there is a 'Parse JSON' block connected to an 'http:// Petición GET' block. The URL is 'https://api.openweathermap.org/data/2.5/weather?...', and the BasicAuth fields are empty. This is followed by 'Enviar "Lugar:"', 'Enviar JSON Get key { "name" } as Text', 'Enviar "Velocidad del viento:"', 'Establecer datos viento = JSON Get key { "wind" } as Text', 'Parse JSON datos viento', and finally 'Enviar JSON Get key { "speed" } as Text'.

Resultado en la consola:

### ArduinoBlocks :: Consola serie

---

Baudrate:

---

OpenWeather wind info:

Lugar: Ibi

Velocidad del viento:

4.03

## ANEXO:

Listado de APIs con datos en formato JSON

<https://apipheny.io/free-api/#apis-without-key>

<https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON>

<https://www.luisllamas.es/arduino-json/>

<https://arduinojson.org/>