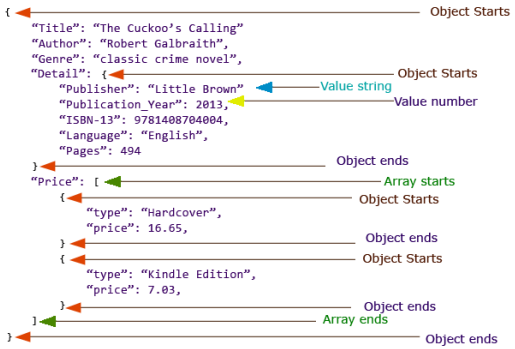
JSON

JavaScript Object Notation (JSON) est un format de données textuelles facilitant l'échange de données structurées entre **tous** les langages de programmation.

Basic Constructs :



① There **four basic and built-in data types** in JSON :

- ✔ strings, numbers, booleans (i.e true and false) and null.

② There are **two structured data types** :

- ✔ **Objects** are a list of label-value pairs : Objects are wrapped within '' and ''.
- ✔ **Arrays** are list of values : Arrays are enclosed by '[' and ']'.

Both objects and arrays can be nested.

L'idée n'est pas ici de refaire un cours sur JSON car le WEB fourmille de choses très intéressantes :

- ✔ https://www.w3resource.com/JSON/introduction.php
- ✔ https://la-cascade.io/json-pour-les-debutants/
- ✔ ...

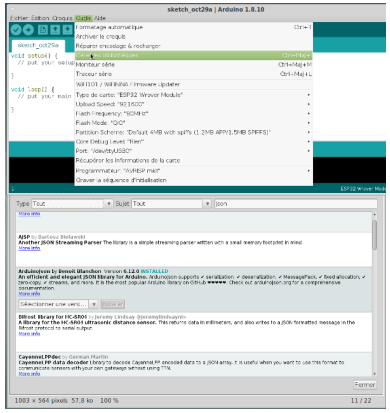
Par contre, on peut s'attarder sur l'aspect "utilisation dans un contexte ESP".

```
{
  "SensorType" : "Temperature",
  "Value" : 21,
  "Hist" : {"Cnt" : 3, "Last" : [20,21,23]},
  "Tol"  : true
}
```


Choix d'une API JSON sur l'ESP ?

On prend la bibliothèque "ArduinoJson" de B.Blanchon :

- ✔ https://github.com/bblanchon/ArduinoJson



Dans l'environnement Arduino IDE, vous installez cette bibliothèque par le menu de "Outils/Gérer les bibiliothèques".

Des exemples : Coté ESP32

Vous trouvez facilement des exemples d'utilisation de cette API

- ✔ https://arduinojson.org/v6/example/
- ✔ https://techtutorialsx.com/2017/04/27/esp32-creating-json-message/

et des conseils d'utilisation :

- ➢ Sensibilisation à l'occupation mémoire des objets JSON :

  "Use **String** objects sparingly, because ArduinoJson duplicates them in the JsonObject. **Prefer plain old char[]**, as they are more efficient in term of code size, speed, and memory usage".

- ✔ https://techtutorialsx.com/2019/05/02/esp32-arduinojson-v6-serializing-json/

Je vous propose "ma" contribution "GPS data" ... à essayer !

```
{
  "sensor":"gps",
  "time":1351824120,
  "data":[48.756080,2.302038]
}
```

```
/*
 *  Simulation d'une payload issue d'un capteur GPS :
 *
 *  SERIALISATION
 *        |
 *        v
 * EMISSION de payload
 *        |
 *        v
 * RECEPTION DE payload
 *        |
 *        v
 * DESERIALISATION !
 *
 * Fichier : esp_json.ino
 * Modifie par G.MENEZ
 */
#include <ArduinoJson.h>

void setup() {
  Serial.begin(9600);
  Serial.println();
}

void loop() {
  Serial.println("==========");
  /*
   * SERIALISATION : On construit un doc JSON que l'on serialise
                     pour fabriquer le PAYLOAD
   */
  StaticJsonDocument<256> jdoc; //Why static ?
                                //=> https://arduinojson.org/v6/api/staticjsondocument/
  jdoc["sensor"] = "gps";
  jdoc["time"]   = 1351824120;
  JsonArray data = jdoc.createNestedArray("data");
  data.add(48.756080);
  data.add(2.302038);

```



```
  Serial.println("Serialization of the String before Emission => we make the JSON Doc : ");
  //serializeJson(jdoc, Serial); // This one serializes DIRECTLY in Serial
  // => so prints on console {"sensor":"gps","time":1351824120,"data":[48.756080,2.302038]}
  char payload[256];
  serializeJson(jdoc, payload); // This one put the serialized Json Document in a STRING
  // payload <= "{\"sensor\":\"gps\",\"time\":1351824120,\"data\":[48.756080,2.302038]}";
  Serial.println(payload);

  /* Si on devait envoyer la payload ca serait la et maintenant !    */

  /* On peut imaginer que la requete serait suivie d'une reponse
     et que l'on vient de la recuperer dans "payload". */

  /*
   * DESERIALISATION : A partir la string recue dans le PAYLOAD
   * on construit un doc JSON
   */
  StaticJsonDocument<256> jdoc_new; /* qui donne acces aux elements      */
  deserializeJson(jdoc_new, payload);
  Serial.println();
  Serial.println("Reception and Unserialization of the String => we use the JSON Doc : ");
  const char* sensor = jdoc_new["sensor"];
  long time          = jdoc_new["time"];
  double latitude    = jdoc_new["data"][0];
  double longitude   = jdoc_new["data"][1];
  Serial.print("Sensor : "); Serial.println(sensor);
  Serial.print("Time : "); Serial.println(time);
  Serial.print("Latitude : "); Serial.println(latitude);
  Serial.print("Longitude : "); Serial.println(longitude);
  Serial.println();
  delay(10000);
}
```


API : Coté "Serveur"

Tous les langages Java, Python, Javascript . . . proposent une API JSON.

En Python :

```
import json
#=== DESERIALISATION ========
# Received Payload : some JSON formatted string
recv_payload =  '{"sensor":"gps", \
                  "time":1351824120,\
                  "data":[48.756080,2.302038]}'
print(recv_payload)

# Parse and Work With ...
d = json.loads(recv_payload)
# the result is a Python dictionary :
print(d["data"])
d["time"] = 0

#=== SERIALISATION ========
# Convert dict to JSON
sent_payload = json.dumps(d) # the result is a JSON string
# ... READY TO BE SENT !!

nice = json.dumps(d, indent=2, sort_keys=True, separators=(", ", ": "))
print(nice)
```


L'API JSON Python est riche :

https://docs.python.org/fr/3/library/json.html

Lorsque l'on convertit de Python en JSON, les objets suivants sont utilisés (et mis en correspondance) :

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

Compte tenu de la simplicité de ce dont on besoin, on pourrait aussi utiliser une solution plus légère (MAIS plus rapide !) : `simplejason`

https://stackoverflow.com/questions/712791/

what-are-the-differences-between-json-and-simplejson-python-modules

Rich MQTT topics / Rich Payload

Quel choix de conception pour un UseCase "classique" :

➢ "Vous devez transmettre les données d'une dizaine d'Objets ESP contenant des capteurs de température, lumière, humidité, ..."

Comment s'y prendre ?

➢ Un topic MQTT par capteur ? et sa valeur en payload ?

House/Sensor1/Temp 21
House/Sensor1/Light True
House/Sensor2/Temp 18 ...

➢ ou un topic par objet et des valeurs de capteurs en payload ?

```
/House/Sensor1 {"Temp": 21, "Hum": 110, Light: true}
```

➢ ou un topic pour la maison ?

```
/House {"Sensor1": {"Temp": 21, "Hum": 110, Light: true},
        "Sensor2": {...
       }
```


Plus de topics ?

➢ Mais aussi plus de messages (charge réseau) ... moins bien.

➢ Un "asynchronisme" dans la collecte ... plutôt bien.

➢ ...

Moins de topics ?

➢ Moins de messages ... plutôt bien.

➢ Un "synchronisme" dans la collecte ... moins bien.

➢ La possibilité de crypter le payload ... plutôt bien.

Une réflexion ! ... selon le cahier des charges.

Crypter le message : AES-128-CBC

```
1  from cryptography.fernet import Fernet
2  #--- Encryption engine
3  #Generation dynamique d'une cle
4  cipher_key = Fernet.generate_key()
5  print(cipher_key)
6
7  # On peut aussi a prtir de l'exemple en creer une et la partager.
8  cipher_key=b'p7lqIMC18kWn64cZFGe9lZvl4iv4UQeG41miCFLwRQc='
9  cipher = Fernet(cipher_key)
10
11 #----Original
12 payload = b'on'
13 print("\nOriginal message =", payload)
14 #----Encryption
15 encrypted_payload = cipher.encrypt(payload)
16 print("\nEncrypted message =", encrypted_payload)
17 #---------- Transmission ----------
18 #----Decryption
19 decrypted_payload = cipher.decrypt(encrypted_payload)
20 print("\nreceived message =", str(decrypted_payload.decode("utf-8")))
```

Coté ESP, il y aussi des bibliothèques de cryptage :

➢ **Neutralité en terme de consommation** ?