le format JSON

1. Présentation de JSON :

1.1. Définition

JavaScript Object Notation (JSON) est un format de données populaire, dérivé de la notation des objets du langage JavaScript, utilisé pour représenter des données structurées. Ces structures de données sont universelles puisque pratiquement tous les langages de programmation modernes les proposent sous une forme ou une autre. Ces propriétés font de JSON un format d'échange de données idéal. Il est courant de transmettre et de recevoir des données entre un serveur et une application Web au format JSON.

1.2. Normalisation

Créé par Douglas Crockford entre 2002 et 2005, la première norme du JSON est ECMA-404 qui a été publiée en octobre 2003.
Il est actuellement décrit par les deux normes en concurrence :

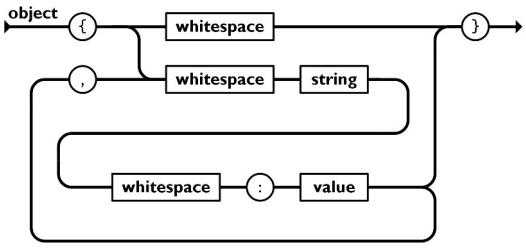
- RFC 82593 de l'IETF,
- et ECMA-4044 de l'ECMA.

La dernière version des spécifications du format date de décembre 2017.

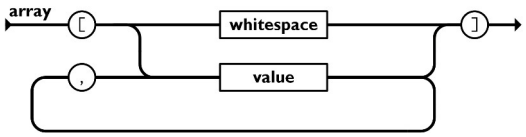
2. Les structures JSON :

Le format JSON (cf. https://www.json.org/json-fr.html) comprend deux types d'éléments structurels principaux :

- le **JSON Object** : un ensemble (collection) de couples "nom/valeur"

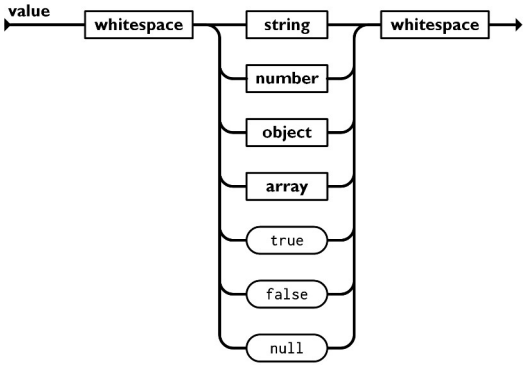


- le **JSON Array** : une liste de valeurs ordonnées



Ces éléments structurels peuvent être imbriqués pour obtenir des structures plus ou moins complexes.

Les données JSON ou **JSON Value** représentées sous le terme "value" sur les représentations structurelles peuvent prendre différentes forme:



3. JSON & Python :

Le module Python "json" permet de travailler avec JSON (chaîne ou fichier contenant un objet JSON). Vous pouvez consulter les possibilités offertes par le module sur le lien : https://docs.python.org/fr/3/library/json.html

3.1. Correspondance : objet JSON - objet Python

JSON	Python
Object	dict
Array	list
	tuple

JSON	Python
String	str
Number	int
	float

JSON	Python
true	True
false	False
null	None

3.2. Sérialisation (conversion Python => JSON)

Le processus d'encodage JSON est généralement appelé sérialisation. Ce terme fait référence à la transformation de données en une série d'octets à stocker ou à transmettre sur un réseau.

Le module "json" propose 2 méthodes de sérialisation :

- dump() : conversion d'une structure Python vers une structure JSON (dans un fichier JSON),
- dumps(): conversion d'une structure Python vers une structure JSON (sous forme de chaine de caractère).

```
import json
dataPython = {"lastName": "Crockford",
              "firstName": "Douglas" }
# vers un fichier JSON
with open("dataJSON.json", "w") as acces_ecriture:
      json.dump(dataPython, acces_ecriture)
# vers une structure JSON (sous forme de chaine de caractère)
dataJSON = json.dumps(dataPython)
```

3.3. Désérialisation (convertion JSON => Python)

la désérialisation est le processus réciproque de décodage des données qui ont été stockées ou livrées dans le standard JSON.

Le module "json" propose 2 méthodes de désérialisation :

- load() : conversion d'une structure JSON (dans un fichier JSON) vers une structure Python,
- loads(): conversion d'une structure JSON (sous forme de chaine de caractère) vers une structure Python.

```
import json
# à partir d'un fichier JSON
with open("dataJSON.json", "r") as acces_lecture:
      dataPython = json.load(acces_lecture)
# à partir d'une structure JSON (sous forme de chaine de caractère)
dataJSON = ' { "lastName": "Crockford", "firstName": "Douglas" } '
dataPython = json.loads(dataJSON)
```