JSON and YAML

JSON

- JSON (JavaScript Object Notation) is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition
- JSON is a way of storing and communicating data with specific rules (like XML, YAML, etc.)
- JSON files has extension .json
- JSON uses key-value pairs
- JSON was designed to be human and machine readable
- JSON is easy to read and write
- Language independent even if it comes from JavaScript

JSON Example

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
      "Catherine",
      "Thomas",
      "Trevor"
  ],
  "spouse": null
}
```

Source: https://en.wikipedia.org/wiki/JSON#Syntax

JSON Data Types (1)

- There are **only 6 data types** in JSON
  - String
  - Number
  - Boolean
  - Null
  - Array
  - Object
- A *value* can be a string, a number, a boolean, a null, an array or an object
- Array and Objects are also called structures
- In a .json file there can be one (and only one) value

JSON Data Types (2)

- String

```
"Hello World"
```

  - It is a sequence of zero or more Unicode characters
  - It is wrapped in **double quotes**, using backslash for escape sequences

    `\"` `\\` `\/`

    `\b` (backspace), `\f` (formfeed), `\n` (linefeed), `\r` (return), `\t` (tab)

    `\u` (4 hex digits UNICODE)
  - A character is represented as a single character string

- Number

```
10   1.5   30   1.2e10
```

  - Any kind of number integer, float, exponential
  - The octal and hexadecimal formats are not supported

JSON Data Types (3)

- Boolean
  - Values can be

```
true    false
```

- Null
  - Value can be

```
null
```

JSON Data Types (4)

- Array

```
[ 1, 2, 3]   [ "Hello", "World" ]
```

  - It is an ordered list of values
  - It begins with a left square bracket `[` and ends with a right square bracket `]`
  - Values are separated by comma `,`
  - It is not mandatory all values have the same data type

```
[ 1, "Bye", true]
```

- Object

```
{ "name" : "John" }   { "age" : 21 }
```

  - It is an unordered set of name-values pairs
  - It begins with a left curly brace `{` and ends with a right curly brace `}`
  - Each name is followed by colon `:`
  - The name-value pairs are separated by comma `,`

JSON Example... again

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
      "Catherine",
      "Thomas",
      "Trevor"
  ],
  "spouse": null
}
```

JSON Example... again

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
      "Catherine",
      "Thomas",
      "Trevor"
  ],
  "spouse": null
}
```

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "streetAddress": "21 2nd Street",
  "city": "New York",
  "state": "NY",
  "postalCode": "10021-3100",
  "homePhoneNumber": "212 555-1234",
  "officePhoneNumber": "646 555-4567",
  "child1": "Catherine",
  "child2": "Thomas",
  "child3": "Trevor",
  "spouse": null
}
```

JSON-LD (JSON for Linked Data)

JSON-LD is a lightweight Linked Data format. It is a method of encoding Linked Data using JSON. It is designed to provide a way to help JSON data interoperate at Web-scale by adding **semantic context** to existing structures.

- **@context**: Maps terms to IRIs (Internationalized Resource Identifiers), defining what the keys in the JSON represent.
- **@id**: Uniquely identifies the resource (using a URI/IRI).
- **@type**: Specifies the type of the object (e.g., a Person, a Dataset, or a Publication).

Definition in JSON-LD:

```
{
  "@context": "[https://schema.org/](https://schema.org/)",
  "@id": "[https://orcid.org/0000-0002-1825-0097](https://orcid.org/0000-0002-1825-0097)",
  "@type": "Person",
  "name": "John Smith",
  "jobTitle": "Data Manager"
}
```

JSON vs. JSON-LD: Adding Meaning

Standard JSON (No Context) The keys are local and ambiguous. A machine doesn't know what "name" refers to globally.

```
{
  "name": "Advanced Data Management",
  "code": "ADM-01",
  "teacher": "John Smith"
}
```

JSON-LD (Semantic Context) The @context maps local keys to global definitions (e.g., Schema.org).

```
{
  "@context": "[https://schema.org/](https://schema.org/)",
  "@type": "Course",
  "name": "Advanced Data Management",
  "courseCode": "ADM-01",
  "instructor": {
    "@type": "Person",
    "@id": "[https://orcid.org/0000-0002-1825-0097](https://orcid.org/0000-0002-1825-0097)",
    "name": "John Smith"
  }
}
```

The Difference: JSON-LD enables Interoperability. Any system in the world can now understand that this is a "Course" and who the "instructor" is by looking up the definitions at schema.org.

YAML

YAML (YAML Ain't Markup Language) is a human-readable data serialization standard.
It is widely used for configuration files, data exchange, and more.
YAML files has extension .yaml or .yml

- **Simple and Readable**: Easy to understand and write.
- **Hierarchical Data Structure**: Supports nested structures using indentation.
- **Data Types**: Strings, numbers, booleans, lists, mappings, and more.
- **Cross-Language Compatibility**: Used in many programming languages.

YAML Example

```yaml
---
firstName: John
lastName: Smith
isAlive: true
age: 27
address:
  streetAddress: 21 2nd Street
  city: New York
  state: NY
  postalCode: 10021-3100
phoneNumbers:
- type: home
  number: 212 555-1234
- type: office
  number: 646 555-4567
children:
- Catherine
- Thomas
- Trevor
spouse:
```

YAML Data Types

1. Strings

Strings in YAML can be written in various ways:

- **Unquoted**: Simple strings do not require quotes.
- **Double-quoted (`"`)**: Used for strings containing special characters.
- **Single-quoted (`'`)**: Used to prevent interpretation of special characters.

Examples:

```
name: Mario Rossi
quote: "I thought: 'This is a quote'"
text_with_spaces: 'This is a text with spaces   preserved'
```

YAML Data Types

2. Numbers

YAML supports different types of numbers:

- **Integers**
- **Floats**
- **Exponentials**
- **Hexadecimal**

Examples:

```yaml
integer: 42
float: 3.14
exponential: 1.6e-35
hexadecimal: 0x1A
```

YAML Data Types

3. Booleans

Boolean values in YAML are represented by the keywords `true` or `false` (case-sensitive).

Examples:

```yaml
enabled: true
disabled: false
```

> **Note**: Avoid using ambiguous words like `yes`, `no`, `on`, `off`, as they may be interpreted as booleans in some YAML implementations.

YAML Data Types

4. Null

The `null` value can be represented with keywords such as `null`, `~`, or by leaving the field empty.

Examples:

```yaml
null_value: null
short_null: ~
empty_null:
```

YAML Data Types

5. Lists (Arrays)

Lists in YAML are indicated by the `-` character for each element.

```yaml
fruits:
  - Apple
  - Banana
  - Orange

numbers:
  - 1
  - 2
  - 3
```

You can also nest lists:

```yaml
menu:
  - name: Starter
    options:
      - Pasta
      - Risotto
  - name: Main Course
    options:
      - Meat
      - Fish
```

6. Mappings (Dictionaries/Hashes)

Mappings in YAML are key-value pairs separated by `:`. They are similar to dictionaries in Python or objects in JavaScript.

Examples:

```yaml
user:
  name: Mario Rossi
  age: 30
  city: Rome

configuration:
  database:
    host: localhost
    port: 3306
    username: admin
    password: secreto
```

7. Multi-line (Block Literals and Folded Strings)

YAML offers two ways to handle multi-line strings:

- **Block literals (`|`)**: Preserves spaces and newlines.
- **Folded strings (`>`)**: Compresses newlines into a single space.

```yaml
literal_description: |
  This is a description
  on multiple lines.
  Newlines are preserved.

folded_description: >
  This is a description
  on multiple lines.
  Newlines are compressed into a single space.
```

Output:

```
This is a description
on multiple lines.
Newlines are preserved.

This is a description on multiple lines. Newlines are compressed into a single space.
```

YAML Data Types

8. Timestamps

YAML supports date and time representation in ISO 8601 format.

Examples:

```yaml
date: 2023-10-15
time: 14:30:00
timestamp: 2023-10-15T14:30:00Z
```

9. Complex Types

YAML allows you to define complex types through references and aliases.

Examples:

- **Aliases**: Allows reusing an existing node without duplicating it.

```yaml
name: &common_name Mario Rossi
user1:
  name: *common_name
  age: 30
user2:
  name: *common_name
  age: 25
```

- **Explicit Tags**: You can explicitly specify the type of a value using tags.

```yaml
integer_number: !!int 42
float_number: !!float 3.14
boolean_value: !!bool yes
```

10. Comments

Although not a data type, comments in YAML start with `#` and are ignored during parsing.

Example:

```yaml
name: Mario Rossi  # User's name
age: 30            # User's age
```

Summary of Data Types in YAML

Type	Description	Example
Strings	Regular text or quoted strings.	`name: Mario Rossi`
Numbers	Integers, floats, exponentials, hexadecimal numbers.	`pi: 3.14`
Booleans	Values `true` or `false`.	`enabled: true`
Null	Represents the absence of a value.	`value: null`
Lists	Ordered collection of elements.	`- item1` `- item2`
Mappings	Key-value pairs.	`key: value`
Multi-line	Multi-line text with `	`(literals) or` >` (folded).
Timestamps	Dates and times in ISO 8601 format.	`date: 2023-10-15`
Aliases	Reuse of existing nodes.	`&alias` and `*alias`
Explicit Tags	Explicit specification of data types.	`!!int 42`

JSON vs YAML

JSON (JavaScript Object Notation)

- **Syntax**: Strict, uses braces `{}` and brackets `[]`.
- **Data Types**: Strings, numbers, booleans, arrays, objects (no dates natively).
- **Readability**: Machine-friendly, less human-readable due to quotes and commas.
- **Use Cases**: APIs, web applications, data interchange.

YAML (YAML Ain't Markup Language)

- **Syntax**: Flexible, indentation-based, no braces or brackets.
- **Data Types**: Strings, numbers, booleans, lists, mappings, dates, and more.
- **Readability**: Human-friendly, cleaner syntax without quotes or commas.
- **Use Cases**: Configuration files, data serialization, documentation.

Example Comparison

JSON Example

```json
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "courses": ["Math", "Physics"]
}
```

YAML Example

```yaml
name: John Doe
age: 30
isStudent: false
courses:
  - Math
  - Physics
```

XML vs (JSON or YAML)

- All can store data (and metadata) to transfer them
- All are language-independent
- All have hierarchical structure
- JSON/YAML are simpler to learn, read and write with respect to XML
- JSON/YAML files are more human-readable than XML files
- JSON supports few data types with respect to XML
- XML supports include and import of external XML or binary files
- XML supports references (entities, etc.)
- XML supports attributes
- XML supports namespaces
- XML has a robust schema document (XSD) format for validation process
- JSON/YAML validation schema is more a “well formed” check than a proper validation

XML vs JSON

XML fragment:

```xml
<pixel_reading>
      <device>Camera</device>
      <patch>cyan</patch>
      <RGB resolution="8">
            <red>0</red>
            <green>255</green>
            <blue>255</blue>
      </RGB>
</pixel_reading>
```

JSON:

```json
{
  "pixel_reading": {
    "device": "Camera",
    "patch": "cyan",
    "RGB": {
      "reading": {
        "red": 0,
        "green": 255,
        "blue": 255,
      }
    }
  }
}
```

⇐ The resolution attribute is missing!

XML vs JSON

XML fragment:

```xml
<pixel_reading>
      <device>Camera</device>
      <patch>cyan</patch>
      <RGB resolution="8">
            <red>0</red>
            <green>255</green>
            <blue>255</blue>
      </RGB>
</pixel_reading>
```

JSON:

```json
{
  "pixel_reading": {
    "device": "Camera",
    "patch": "cyan",
    "RGB": {
      "resolution": 8,
      "reading": {
        "red": 0,
        "green": 255,
        "blue": 255,
      }
    }
  }
}
```

⇐ To not miss the information, you need to add resolution as child!

JSON and YAML Web Resources

- https://www.json.org
- https://quickref.me/json.html
- https://json-schema.org/
- https://www.jsonschemavalidator.net
- https://www.w3schools.com/python/python_json.asp
- https://json-ld.org/
- https://schema.org/
- https://yaml.org
- https://quickref.me/yaml.html
- https://www.yamllint.com
- https://www.bairesdev.com/tools/json2yaml/