1. JSON, the lightweight language

1.1. Introduction

Learning objectives, concepts and skills:

- Identify the general features of JSON format.
- Express the structure and syntax in this language.

1.2. Why JSON?

The term "markup language" encompasses a broad set of languages. Each of them has its own rules and characteristics.

JSON is different, mainly because of its simplicity. It has no tags and it is defined by a small, compact syntax capable of representing hierarchical data structures using a small set of data types.

Thanks to these characteristics, JSON is a language widely used in multiple environments like configuration files, data exchange between applications, and especially in web application development.

JSON has been standardized by ECMA International and the **IETF** (Internet Engineering Task Force).

Some of the **factors** that have popularized JSON are:

- The explosive growth of RESTful APIs based on JSON.
- The simplicity of JSON's basic data structures.
- The increasing popularity of JavaScript.

1.3. Introduction to JSON

JSON is the acronym for **JavaScript Object Notation**. It is a lightweight text format that allows data to be exchanged. **JSON** is a **metalanguage** used to create sections of text in a format that can be easily read, parsed and interpreted.

JSON is based on a subset of **JavaScript**, an imperative, object-oriented, interpreted programming language created by Brendan Eich. However, it maintains independence from this language.

JSON originally emerged as an alternative to XML (eXtensible Markup Language), a metalanguage. Because of its fast readability and smaller size, JSON gained rapid acceptance. **JSON** is **simpler** than XML, but **XML** is more **powerful**.

For humans, reading and writing JSON are simple actions. Computers, on the other hand, have no difficulty generating and interpreting it.

That is why it is frequently used for **data transmission** in web applications.

Documents written in this language are extremely easy to process in JavaScript, since it is its natural programming language. It is also very easy to process them in other programming languages in which processors are available, which are practically all modern programming languages.

JSON (or some of its dialects) is used to **store information** in **non-relational databases**.

The same data expressed in JSON consumes **less memory space** than in other languages, making it a more **efficient solution** for storing large volumes of information.

The following two code blocks show the same data represented in XML and JSON languages. As can be seen, the JSON document uses less bytes (114 bytes) than the XML version (145 bytes), approximately 30% less space.

XML document:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <name>Martín</name>
    <surnames>Rubio Piñeiro</surnames>
    <age>22</age>
</person>
```

JSON document:

```json
{
    "person": {
        "name": "Martín",
        "surnames": "Rubio Piñeiro",
        "age": 22
    }
}
```

JSON and **XML** are two most popular file formats for **interchanging data**, but we must keep in mind that they serve different purposes and have different capacities. Both are text-based human-readable formats with well-documented **open standards** on the World Wide Web.

One of the most significant advantages of using JSON is that the file size is smaller; thus, transferring data is faster than XML. Moreover, since JSON is compact and very easy to read, the files look cleaner and more organized without empty tags and data. The simplicity of its structure and **minimal syntax** makes JSON easier to be used and read by humans.

Contrarily, XML is often characterized for its complexity and old-fashioned standard due to the tag structure that makes files bigger and harder to read.

However, JSON vs. XML is not entirely a fair comparison. JSON is often wrongly perceived as a substitute for XML, but while JSON is a great choice to make simple data transfers, it does not perform any processing or computation. **XML** might be "old" and complex, but its complexity is what enables this language to not only transfer data but also to **process** and **format** objects and documents. There are many tools and technologies around XML that do not exist for JSON.

One great advantage of using XML is that it handles **comments**, **metadata**, and **namespaces**. This feature makes it easier for the developer to keep track of what is happening and to share the document with other team members. Moreover, **XML** enables various **data types** (such as images and charts), unlike JSON, which only supports strings, objects, numbers, and Boolean arrays.

The way data is store in XML differs from JSON. While the **markup language** stores data in a **tree structure**, contrarily, **JSON** stores it like a **map**, which entails **key-value pairs**. Moreover, **JSON** does **not** utilize **end tags** and can use **arrays** (data structures with groups of elements).

There are no better or worse languages, but one that is more suitable for each specific use.

1.4. JSON syntax and structure

JSON's simplicity is part of its appeal. JSON uses the **.json** extension when it stands alone. The **JSON** specification does **not** allow for **comments** in the same way that many programming languages do.

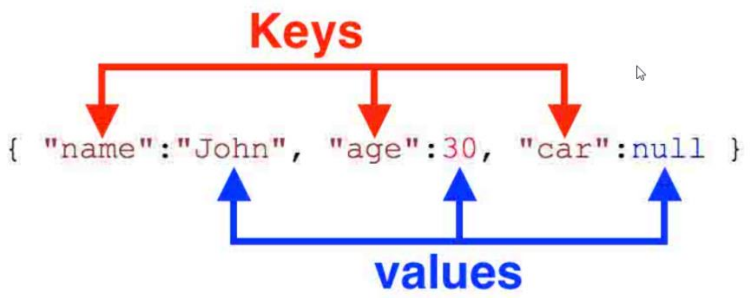
JSON specifies a set of rules for representing structured data in a portable manner. JSON supports **four primitive types** (strings, numbers, Boolean values, and null) and two structured types (**objects** and **arrays**).

An **object** starts and ends with '{' and '}'. Between them, several string value pairs can reside. String and value are separated by a ':' and if there are more than one string value pairs, they are separated by ','. The string value pairs are also referred as key-value pairs.

JSON keys are on the left side of the colon. They need to be wrapped in double quotation marks, as in "key", and can be any valid string. Within each object, keys need to be unique. These key strings can include whitespaces, as in "first name", but that can make it harder to access when you're programming, so it's best to use underscores, as in "first_name".

JSON values are found to the right of the colon. At the granular level, these need to be one of following six data types: strings, numbers, objects, arrays, Booleans (true or false) and null.

In the image we can see a JSON object with its keys and values remarked:



The following is an example of a JSON object:

```
{
   "name": "Angie",
   "age": "42",
   "role": "staff"
}
```

Arrays are sets of data types defined within brackets. So, an array starts and ends with '[' and ']'. Between them, several values can reside. If there are several values, they are separated by ','.

Example of an array of integer:

```
[1, 3, 5]
```

Example of an array of string:

```
["markup","tag","element"]
```

Example of an array with different types of elements:

```
[2, "three", true]
```

The following example shows a complex JSON object:

```
{
  "name":"Alan Turtle",
  "profession":"Programmer",
  "age":25,
  "languages":["PHP","JavaScript","Node.js"],
  "availability_to_travel":true,
  "professional_info": {
      "years_of_experience": 12,
      "level": "Senior"
  }
}
```

The key "languages" in the example has an array of languages as value.

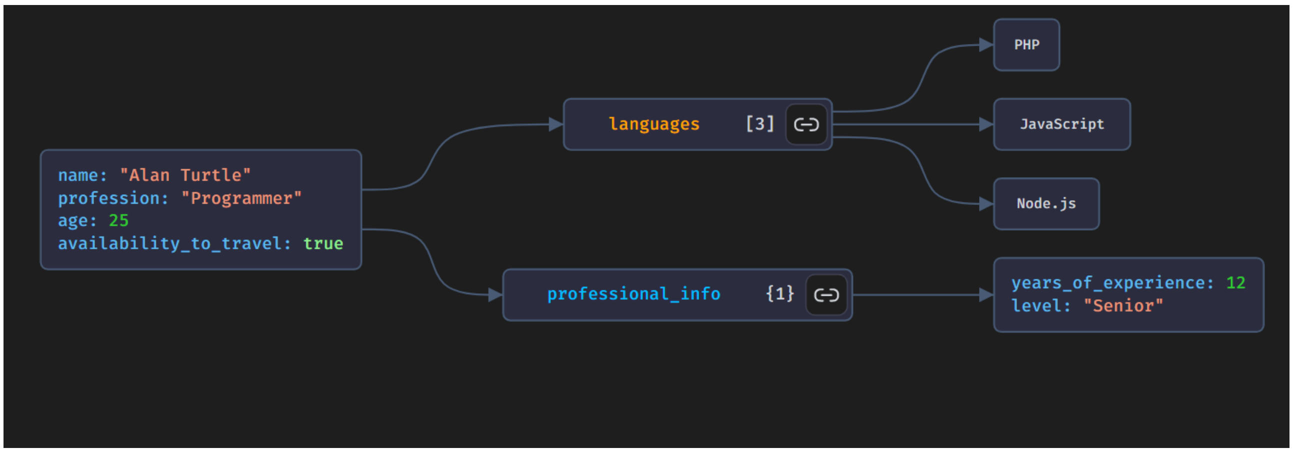
The key "professional_info" in the example has an object as value.

A JSON document is made up of a single element (an object or an array). In both objects and arrays, the last element cannot be followed by a comma.

JSON documents have a tree structure.

We can use https://jsontree.vercel.app/ to convert a JSON document to its tree structure.

The tree structure obtained for the last document seen is:


name: "Alan Turtle"
profession: "Programmer"
age: 25
availability_to_travel: true
languages [3]
PHP
JavaScript
Node.js
professional_info {1}
years_of_experience: 12
level: "Senior"