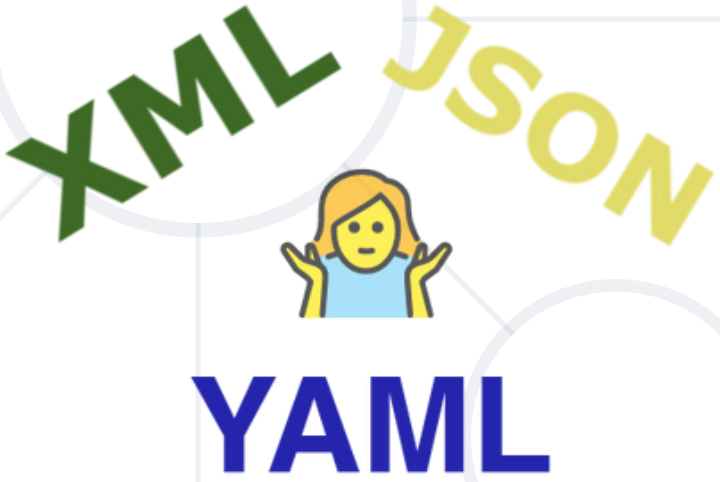
Data Formats

JSON, YAML and XML



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg/

You Have Questions?

Software University

sli.do

#QA-Auto-BackEnd

Table of Contents

Software University

Data Formats

Introduction

Introduction to Data Formats

Software University

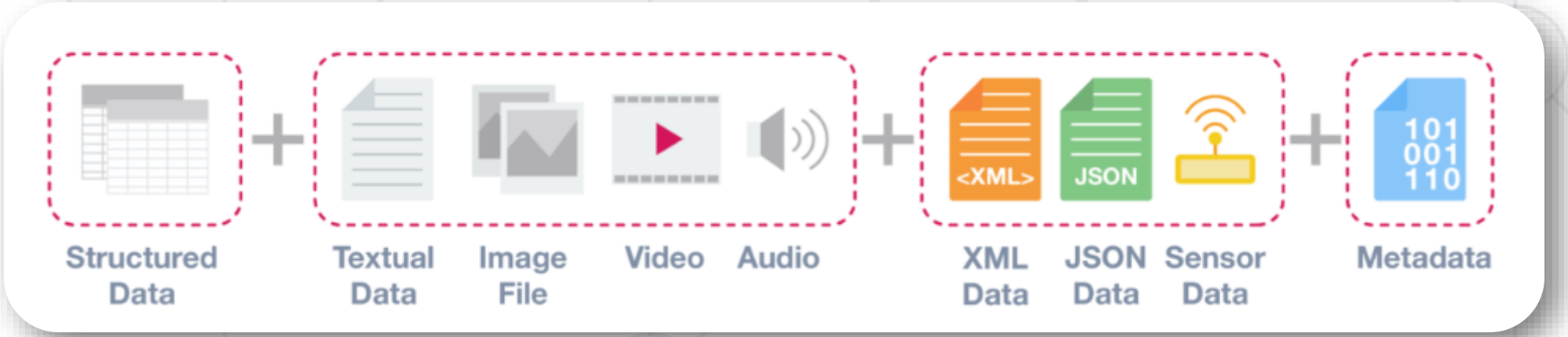
- Standardized **ways** to **structure**, **organize**, and **represent** data
- Ensure **consistent interpretation** and **processing** of data across applications and systems
- **Importance:**
  - Facilitate **data exchange** between applications, databases, and networks
  - Enable **interoperability** and **data integration** across diverse systems
  - Streamline **data storage**, **retrieval**, and **processing**

Common Uses

- **Web APIs**
  - Data exchange between web applications and servers
- **Database Systems**
  - Defining data structures and schemas
- **Networking**
  - Encapsulating and transmitting data packets efficiently
- **Configuration Files**
  - Storing configuration settings for software and systems

Different Data Structures

- **Structured** Data
- **Unstructured** Data
- **Semi-structured** Data
- **Metadata** – Data about Data



Structured Data

- **Well-defined** and organized data
- **Easily analyzed** and **processed** due to its organized structure
- Commonly represented in **tabular formats**
- **Excel files** and **SQL databases** are examples of structured data
- Each field in structured data is **discrete** and can be **accessed individually** or **together**
- Highly **efficient** for **data aggregation** and **retrieval**
- **Traditional form** of data storage, from the early days of databases

Unstructured Data

- Data that **lacks** a **predefined structure** or organization
- **Commonly** represented in **text format**, but may contain **other data types**
- **Difficult** to **analyze** and **process** using traditional methods due to irregularities and ambiguities
- Common examples include **audio**, **video** files, **text documents**, and **NoSQL databases**
- **Advanced technologies** and tools are emerging to handle unstructured data effectively

Structured vs. Unstructured Data

- **Structured data**
  - Organized and well-defined schema
  - Follows a specific format or schema
  - Easier to analyze and process
  - Examples: Database tables, CSV files
- **Unstructured data**
  - No predefined format or organization
  - Can be text, images, audio, videos
  - Requires more complex processing techniques
  - Examples: Social media posts, emails, documents

Semi-Structured Data

- Falls **between** structured and unstructured data
- Does not adhere to the strict structure of relational databases, but **contains tags** or markers to indicate data groupings
- **Self-describing** structure makes it easier to analyze than unstructured data
- Widely **used** in **web applications** and other data-driven environments
- Enables **efficient data analysis** and **processing** without the complexities of traditional structured data

Semi-Structured Data Formats

- **JSON** (**J**ava**S**cript **O**bject **N**otation)
  - Human-readable data-interchange format
- **YAML** (**Y**AML **A**in't **M**arkup **L**anguage)
  - Human-readable data serialization language, offering flexibility and expressivity
- **XML** (e**X**tensible **M**arkup **L**anguage)
  - Standardized markup language, suitable for complex data with nested hierarchies

JSON Data Format

Definition and Syntax

JSON History and Evolution

- **Early Origins:**
  - Developed from **JavaScript**
  - Lighter **XML alternative**
  - Gained popularity for its **ease of use**
- **Widespread Adoption:**
  - Standard for **web data exchange**
  - Popular across **various domains**
  - Established as a **key data format**

Elements of JSON Syntax

- **Curly brackets** ({ }) for enclosing objects
- **Square brackets** ([ ]) for enclosing arrays
- **Colons** (:) to separate keys from values
- **Commas** (,) to separate key-value pairs or array items
- **Double quotes** (") to enclose strings
- Example:

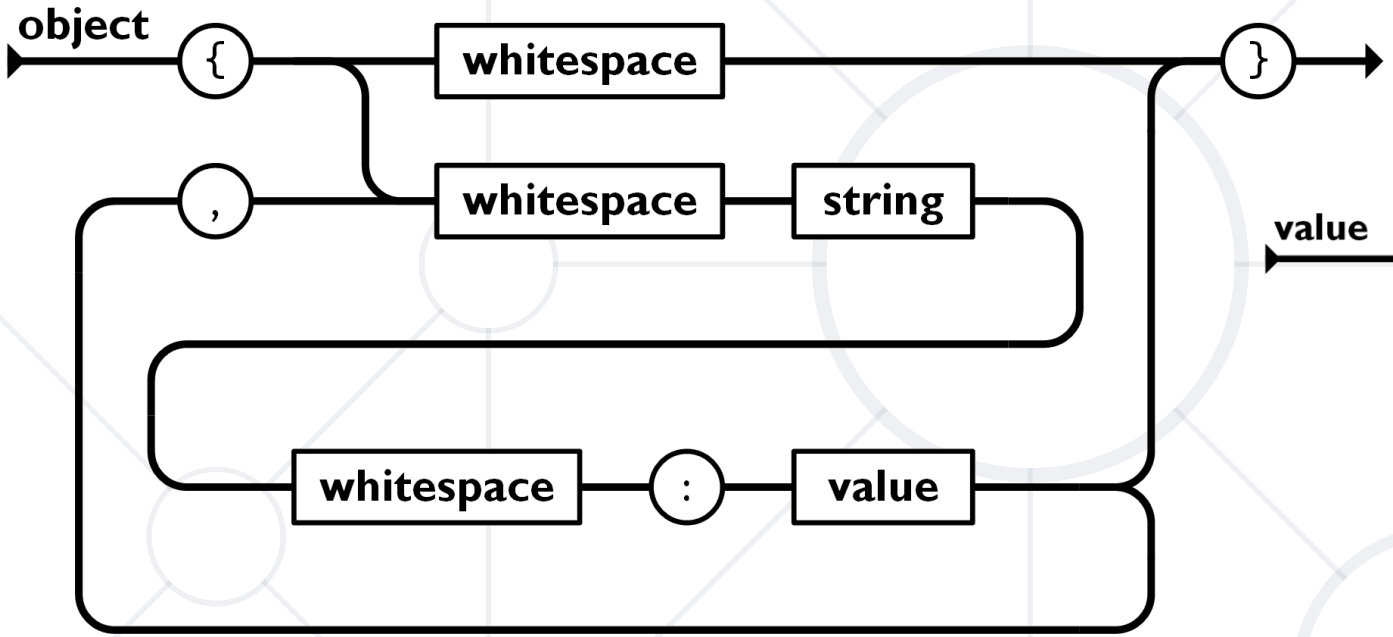
```
//simple object
{
 "name": "John Doe",
 "age": 30,
 "occupation": "Software Engineer"
}
```

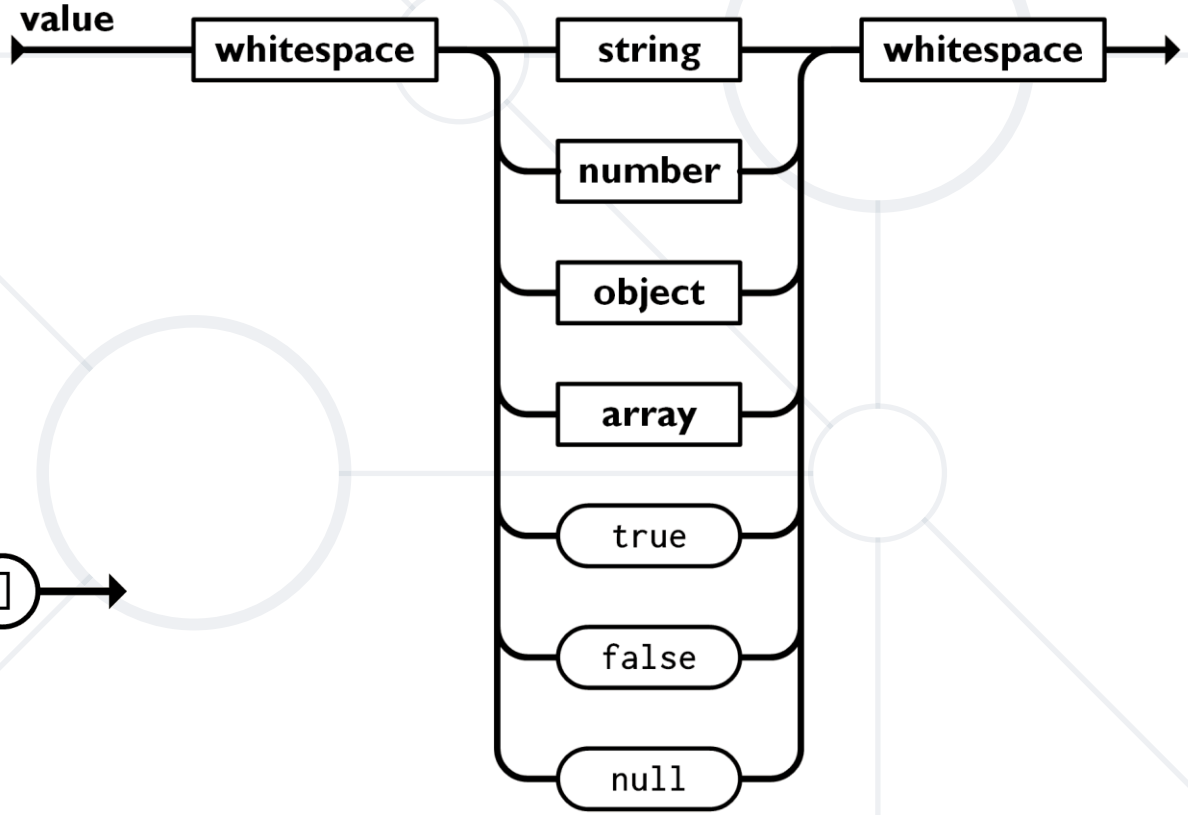
JSON Data Types

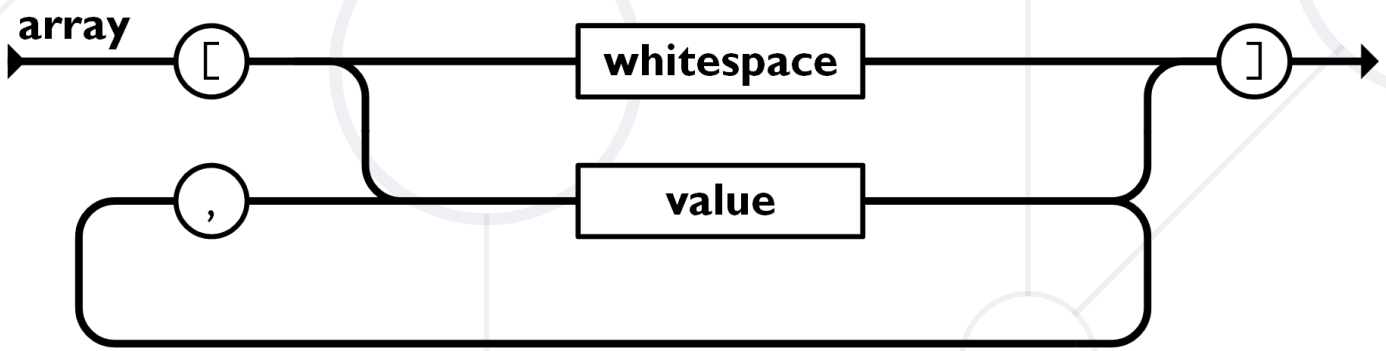
- **Strings**: text data
- **Numbers**: numerical values
- **Booleans**: true or false values
- **Arrays**: collection of values
- **Objects**: collection of key-value pairs
- Example:

```
{
  "name": "Product A", // String
  "price": 100, // Number
  "isAvailable": true, // Boolean
  "items": ["Item1", "Item2"], // Array
}
```

Object, Array and Value in JSON







JSON Example: Simple Data Structure

- A **JSON object** representing a **student's information**:

```
{
 "name": "Alice Smith",
 "age": 25,
 "courses": ["Math", "Science", "English"]
}
```

  - The **object** is enclosed within **curly braces** ({})
  - Each **key-value** pair is separated by a **colon** (:)
  - **String values** are enclosed within **double quotes** (")
  - **Array values** are enclosed within **square brackets** ([])

JSON Example: Nested Data Structure

- JSON object representing a **product** with **multiple properties**:

```
{
 "productID": "Laptop 12345",
 "price": 1200,
 "properties": {
    "processor": "Intel Core i7",
    "memory": 16 GB
  },
  "reviews": [
    {
      "author": "John Bass",
      "comment": "Great product!"
    },
    {
      "author": "Jane Smith",
      "comment": "Satisfied with the performance."
    }
  ]}
```

- Object nested within multiple levels
- Properties object nested within the main object
- Reviews array nested within the main object
- Key-value pair and array element is self-contained and clearly defined

Products Problem

- You are given a table containing information about **4 products**
- **Each product** has the **following attributes**:
  - **Product** (string), **Price** (number), **Description** (string), **Key Words** (an array of strings)
- Your task is to **convert the table** into a **JSON format**

Product	Price	Description	Key Words
Apple	1.50	Fresh green apples	Juicy, Green, Crunchy
Banana	0.30	Ripe yellow bananas	Sweet, Yellow, Soft
Orange Juice	3.00	Freshly squeezed orange juice	Citrus, Vitamin C, Fresh
Chocolate Cake	5.00	Rich and moist chocolate cake	Chocolatey, Rich, Creamy

Products Solution

```
[
    {
        "Product": "Apple",
        "Price": 1.50,
        "Description": "Fresh green apples",
        "KeyWords": ["Juicy", "Green", "Crunchy"]
    },
    {
        "Product": "Banana",
        "Price": 0.30,
        "Description": "Ripe yellow bananas",
        "KeyWords": ["Sweet", "Yellow", "Soft"]
    },
        …
]
```

YAML

Definition and Syntax

YAML History and Evolution

- **Early Origins:**
  - More **readable** and simpler than XML
  - Focused on **clear data** serialization
  - Rose to popularity for being **straightforward**
- **Widespread Adoption:**
  - Common in **configuration** and **system management**
  - Favored in **DevOps** for complex systems
  - Principal format for **deployment** and **setup**

Elements of YAML Syntax

- **Indentation**: **Spaces** are used to denote structure
- **Hyphens** (**-**): Used for creating bullet lists (arrays)
- **Colons** (**:**): Separate keys from values
- **Comments** (**#**): Used to add comments within YAML files
- Example:

```
name: John Doe
age: 30
occupation: Software Engineer
```

- Unlike JSON, YAML **does not** typically **use quotes around strings** unless necessary, and does not require commas to separate items

YAML Key-value pairs (Mappings)

Software University

- A **key-value pair** is YAML's basic building block
- Each item in a YAML document is a member of at least one dictionary
- The key is a string, and the **value** can be **any data type** /**number**, **boolean**, **string**, **null**, **array**, and **object**/

Spaces at the beginning of a row are marked with center dots for a clearer view of the indentation

```
name: Product A     # String
price: 100          # Number
isAavailable: true  # Boolean
items:              # Array
··- Item1
··- Item2
```

Data Types: Numbers and Nulls

- **Numbers** may be **decimal**, **floating**, **exponential**, **octal**, and **hexadecimal** types
- **Don't enclose them** in either a single or double quote

```
decimal : 10
float : 2.5
exponential : 5.0e+12
infinity : .inf
octal : 0o12
hexadecimal : 0xF
```

- You can represent a **null value** using either a **~** or **null**, without enclosing them within any quotes

```
foo: ~
bar: null
```

Data Types: Booleans and Comments

- **Boolean values**: **true** or **false**
- **yes**/**no** and **on**/**off** are also interpreted as booleans, unless enclosed in quotes

```
settings:
  enableNotifications: yes
  enableAutoUpdates: no
  isDarkModeEnabled: yes
  isLoggedIn: false
  isAlarmEnabled: off
  isBluetoothEnabled: on
```

- You can **comment** contents of a YAML file using the **#**

```
# Comment on the first line.
# Comment on the second line.
```

Data Types: Strings

- If your string contains any **special characters** that **need escaping** using \, you must enclose the string within double-quotes

```
signature: "John Williams \nSales Executive \nXYZ Company \tLA"
```

- For strings with special characters that don't require escape sequences, use single quotes instead, which would interpret the string as intended

```
':, {, }, [, ], ,, &, *, #, ?, |, -, <, >, =, !, %, @, `'
```

Single Line Strings

- To write a string in multiple lines for any reason but want the **parser to interpret** it as a **single line**, use **>** and write the string content as an indented block

```
message: >
··this is a normal string
··written on multiple lines
··but needs to be treated as a single line
```

- YAML will add a **newline at the end** of the string
- If you want to prevent this and have **no newline at the end**, you should use **>-**

Multi Line Strings

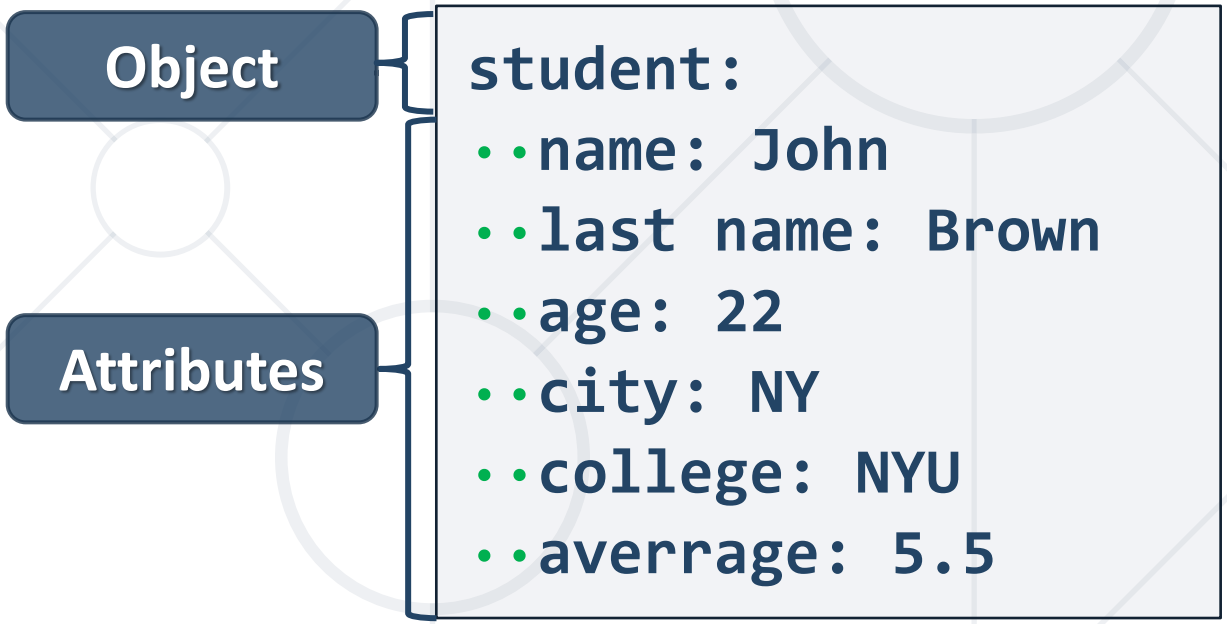
- If you want to write a **string spanning** across **multiple lines**, parsed in the same way as written, use **|**

```
message: |
  this is a multi-line string
  written across multiple lines
  and needs to be treated
  as a multi-line string.
```

- YAML will add a **newline at the end** of the string
- If you want to prevent this and have **no newline at the end**, you should use **| -**

Data Types: Objects

- All the **attributes within an object** should be on the **same level** with the **same indentation** to be part of the same object and to be a valid YAML



Data Types: Arrays

- For **primitive items** in your array, you can represent them as:

```
teams: [Australia, New Zealand, England, India]
```

- For **objects**:
  - Each item in an array is represented with a **hyphen (-)**
  - All the **items** should be on the **same level** with the **same indentation**

Array of objects with name and rank of a team

```
teams:
··- name: Australia
····rank: 3
··- name: New Zealand
····rank: 4
```

YAML Example: Simple Data Structure

- An **YAML object** representing a **student's information**:

```
name: Alice Smith
age: 25
courses: [Math, Science, English]
```

```
name: Alice Smith
age: 25
courses:
  - Math
  - Science
  - English
```

- **Explanation:**
  - **Key-value pairs** are on **new lines** with a colon and space
  - **Strings don't need quotes**
  - **Arrays** are in **square brackets**, with items **separated by commas**
  - Arrays in brackets are more compact representation, but **less common** than the **block style** (**with hyphens**)

YAML Example: Nested Data Structure

- YAML object representing a **product** with **multiple properties**:

```
productID: Laptop 12345
price: 1200
properties:
··processor: Intel Core i7
··memory: 16 GB
reviews:
··- author: John Bass
····comment: Great product!
··- author: Jane Smith
····comment: Good performance.
```

- Multiple **nested levels**
- "properties" nested **inside main object**
- "reviews" array also nested **within main object**
- Clear key-value and array elements

Countries Problem

- You are given a table containing information about **5 countries**
- **Each country** has the **following attributes**:
  - **Name** (string), **Capital** (string), **Population** (number), **Languages** (an array of strings)
- Your task is to **convert the table** into an **YAML format**

Country	Capital	Population (Mil.)	Languages
Switzerland	Bern	8.5	German, French, Italian, Romansh
Canada	Ottawa	38	English, French
Belgium	Brussels	11.5	Dutch, French, German
South Africa	Pretoria	59.3	Zulu, Xhosa, Afrikaans, Others
India	New Delhi	1380	Hindi, Tamil, Telugu, Urdu, Others

Countries Solution

```
- name: Switzerland
··capital: Bern
··population: 8.5
··languages:
····- German
····- French
····- Italian
····- Romansh
- name: Canada
··capital: Ottawa
··population: 38
··languages:
····- English
····- French
# etc.
```

XML

Definition and Syntax

XML History and Evolution

- **Early Origins:**
  - Originated from SGML for **adaptable data structuring**
  - Readable by humans and machines, ideal for **self-defined data**
  - Rapid adoption due to its **document formatting capabilities**
- **Widespread Adoption:**
  - Essential for web services, facilitating system interoperability
  - **Basis** for many **web protocols** and **formats**
  - **Remains effective** for semi-structured data scenarios

XML Specifics

- **Universal notation** (data format / language) for describing structured data using text with tags
- Designed to **store** and **transport** data
- The data is stored together with the **meta-data** about it
- **Tree-like** structure
  - Each element has a start and an end tag
  - Can have attributes and child elements
  - Elements can also have text content
- An XML document has a **root element** that contains all other elements

XML Markup and Content

- An XML document consists of **strings** that:
  - Constitute **markup** – usually begin with **<** and end with **>**
  - Are **content** – placed between markup(**tags**)

person.xml

```
<?xml version="1.0" encoding="UTF-8">
<person>
    <firstName>John Doe</firstName>
</person>
```

Markup tags for Person Object

Content (Person Name)

XML Syntax

- **Header** – defines a **version** and character **encoding**

```
<?xml version="1.0" encoding="UTF-8"?>
```

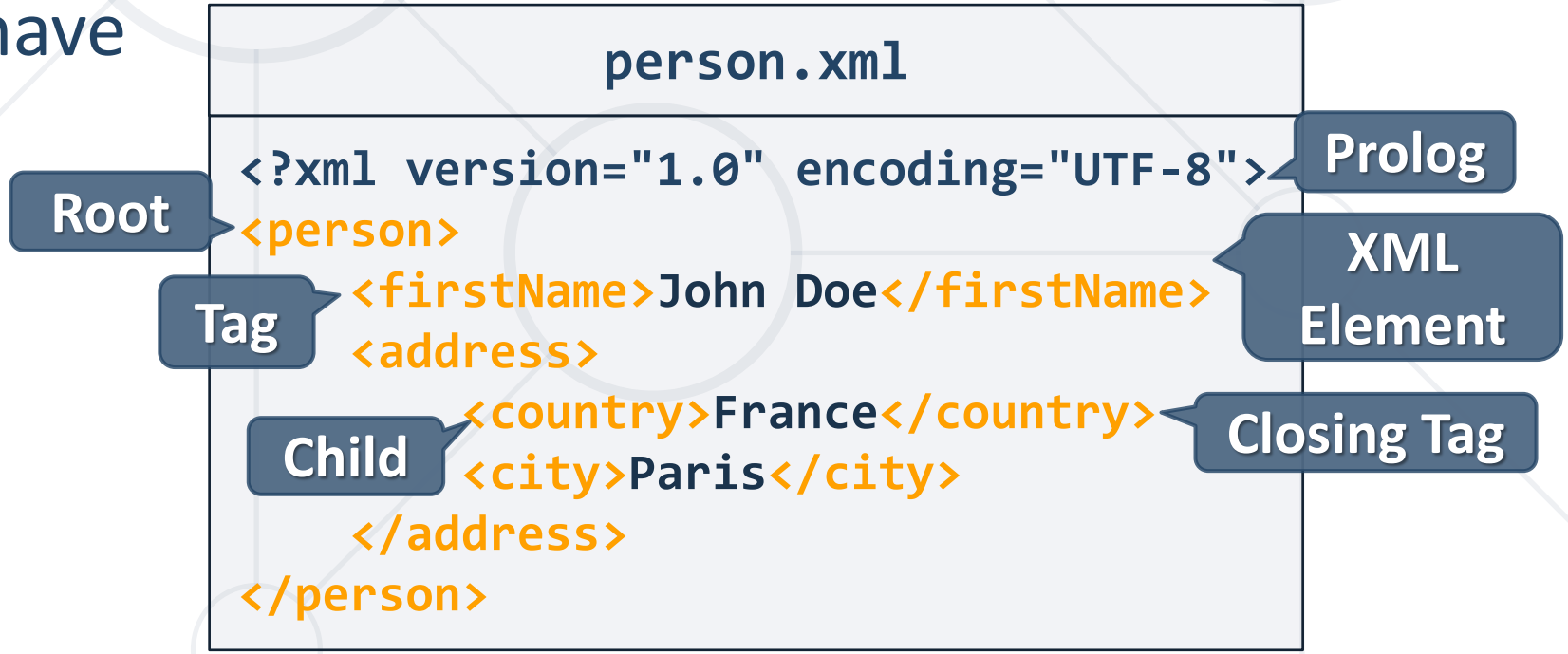
- **Elements** – define the structure
- **Attributes** – element metadata
- **Values** – actual data, that can also be nested elements


```
<title lang="en">Professional C# and .NET</title>
```

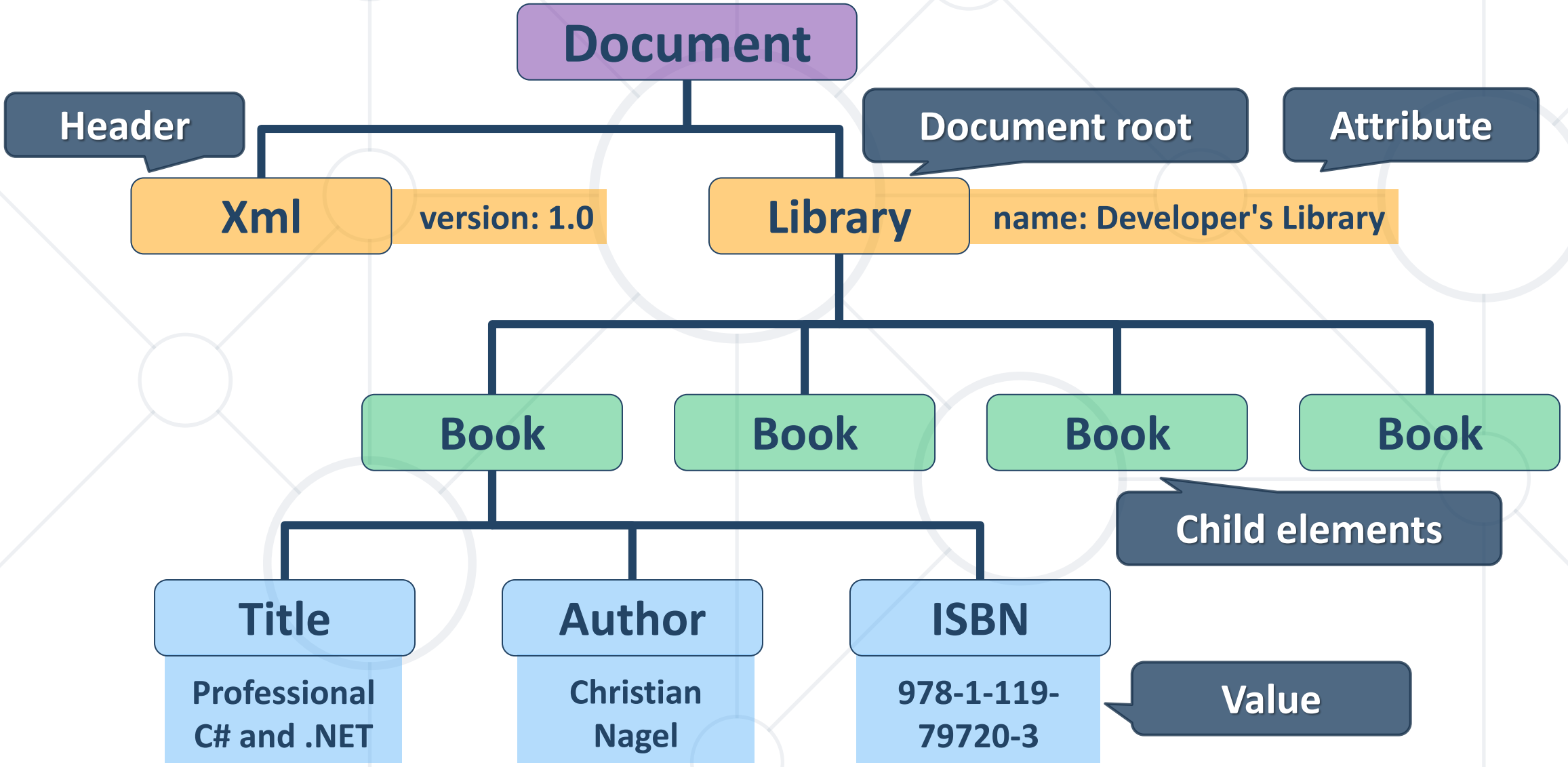
- **Root element** – required to **only** have **one**

XML Structure

- XML documents are formed as **element trees**
- An XML tree starts at a **root element** and branches from the root to **sub elements**
  - All elements can have child elements:



XML Structure



XML Example: Hierarchical Data Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<product>
    <productID>Laptop 12345</productID>
    <price>1200</price>
    <properties>
        <processor>Intel Core i7</processor>
        <memory>16 GB</memory>
    </properties>
    <reviews>
        <review>
            <author>John Bass</author>
            <comment>Great product!</comment>
        </review>
        <review>
            <author>Jane Smith</author>
            <comment>Good performance.</comment>
        </review>
    </reviews>
</product>
```

- **Product** with **multiple properties**:
  - Nested elements structure the document
  - The **"properties" section** is **nested within product**
  - The **"reviews"** are **nested as a list** within product
  - Elements and attributes are well-defined and encapsulated

Cities Problem

- You are given a table containing information about **5 cities**
- **Each city** has the **following attributes**:
  - **Name** (string), **Country** (string), **Population** (number), **Landmarks** (an array of strings)
- Your task is to **convert the table** into a **XML format**

City	Country	Population	Landmarks
Paris	France	2161000	Eiffel Tower, Louvre Museum
Tokyo	Japan	13960000	Tokyo Tower, Sensoji Temple
Cairo	Egypt	9500000	Pyramids of Giza, Egyptian Museum
New York	USA	8419000	Statue of Liberty, Central Park
Rio de Janeiro	Brazil	6748000	Christ the Redeemer, Sugarloaf Mountain

Cities Solution

```
<cities>
  <city>
    <name>Paris</name>
    <country>France</country>
    <population>2161000</population>
    <landmarks>
      <landmark>Eiffel Tower</landmark>
      <landmark>Louvre Museum</landmark>
    </landmarks>
  </city>
  <!- continue with next city-->
<cities>
```

Data Formats Comparison

JSON, YAML, XML Comparison

Software University

Feature	JSON	YAML	XML
Structure	Lightweight	Indentation-based	Hierarchical
Syntax	Braces, Brackets	Indentation, Comments	Tags, Attributes
Readability	High	Higher	Moderate
Use Cases	Web APIs, Config	Config, Settings	Web services, Complex data
Advantages	Simple, Widely supported	Flexible, Readable	Standardized, Complex structure
Disadvantages	Less flexible	Less support	Verbose

JSON, YAML, XML Comparison

Software University

YAML

```yaml
apis:
  - name: login
    port: 8080
  - name: profile
    port: 8090
```

XML

```xml
<apis>
    <api>
        <name>login</name>
        <port>8080</port>
    </api>
    <api>
        <name>profile</name>
        <port>8090</port>
    </api>
</apis>
```

JSON

```json
{
  "apis": [
    {
      "name": "login",
      "port": 8080
    },
    {
      "name": "profile",
      "port": 8090
    }
  ]
}
```

Summary

- Overview of data formats
- Distinctions between structured, unstructured, and semi-structured data
- Popular data formats
  - JSON - lightweight structure and widespread use in web APIs
  - YAML - Flexibility in configuration files
  - XML - hierarchical nature suitable for complex data handling
- Data format comparison - choosing the right format for specific needs

Questions?

SoftUni

Software University | SoftUni Svetlina | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids

SoftUni Diamond Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg