Lab Session 1
XML & JSON

University of Trento
Service Design and Engineering (2019-20)

Team Rocket

Davide Tessarolo   Devis Dal Moro   Mouslim Fatnassi

XML - eXtensible Markup Language

Standard data representation and data **exchange** format
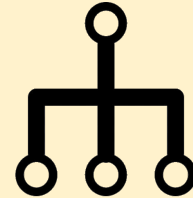
Framework to define markup languages (**meta-language**)

Tree-like structure

Platform independent

(UTF-8 compliant)

e.g. XML for Pokémon data to store them in your Pokédex

XML - Basic concepts

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<!--THIS IS OUR POKEDEX-->
<Team>
    <Pokemon nickname="GreenRipper">
        <species>BULBASAUR</species>
        <dex>4</dex>
        <types>
            <type>GRASS</type>
            <type>POISON</type>
        </types>
        <experience>64</experience>
        <moves>
            <attack>
                <name>RAZOR LEAF</name>
                <type>GRASS</type>
                <power>55</power>
            </attack>
            <effect>
                <name>SLEEP POWDER</name>
                <type>GRASS</type>
            </effect>
        </moves>
    </Pokemon>
</Team>
```

- **Element node**

  (logical grouping of contents)

- **Text node** (actual content)

- **Attribute node** (name value pairs)

- **Comment node**

- **Processing instruction**

XML - Properties

Well Formed

Single root node

Appropriate tag nesting
`<a> <../> </a>`

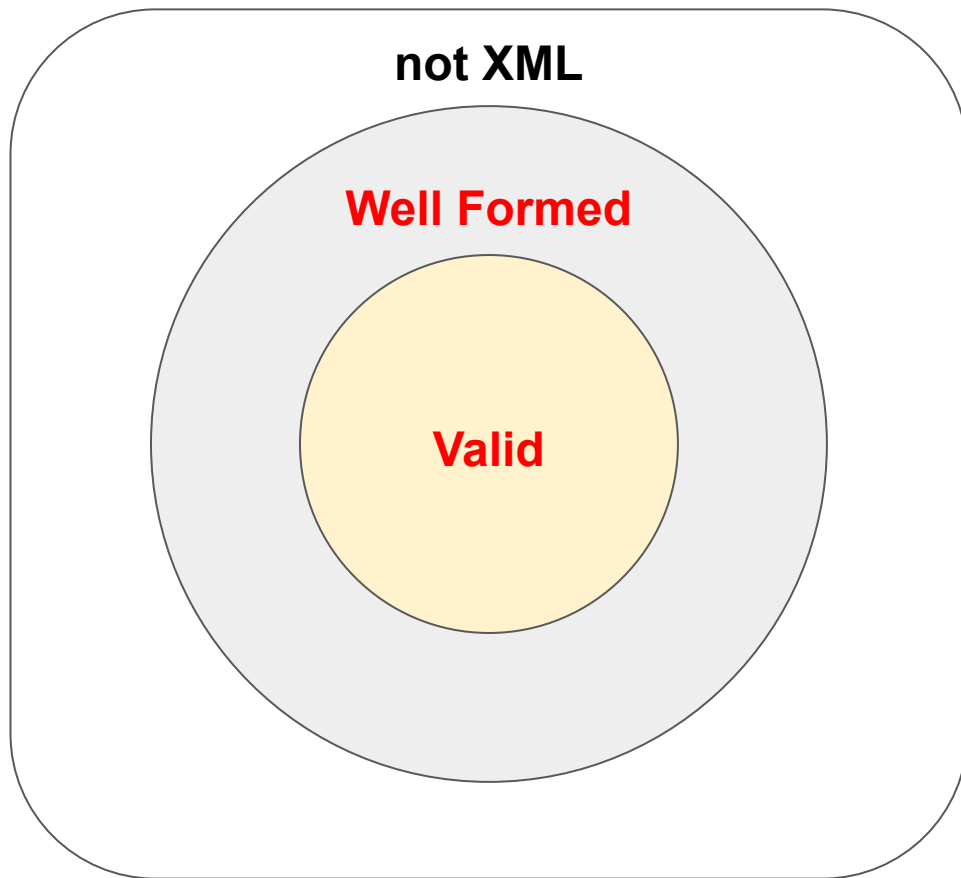No repetition in attributes per element

Validated   **compliance**
with an DTD/XSD (when defined)

XML - Properties

*An XML document **MUST** be well formed*

*If it's **NOT** well formed, it can't be valid either...*

not XML

Well Formed

Valid

XML Schema Definition (XSD)

XML file with a specific grammar to define new XML-based languages and validate its instances

Data types:
by using simple (e.g. string, integer,..) and defining complex (e.g. PokemonType, Move) data types, we can build the **elements** (with their **attributes**) of the XML-based language we're describing

Constraints:
declarative rules for data types and elements (e.g. min/max values and/or occurrences like "*a Pokèmon element must have 1 or two PokémonType assigned to it* ")

Relationships:
express associations between elements (e.g. "*a Pokémon has multiple moves and these could be attacks, effects, ..*")

XSD - Example

- Using the standard xsd schema

- Defining new complex type
  (elements in xml instances of this XML-based lang.)

- using XSD base types
  (string, decimal, integer, boolean, date, time)

- Indicators
  (all, choice, sequence)

- Possible attributes for a Pokémon
  (use="required" if mandatory)

```xsd
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Pokemon">
        <xsd:complexType>
            <xsd:all>
                <xsd:element name="species" type="xsd:string"/>

                <xsd:element name="types">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="type" type="xsd:string"
                                         minOccurs="1" maxOccurs="2"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>

                <xsd:element name="dex" type="xsd:string"/>

                <xsd:element name="experience">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:integer">
                            <xsd:minInclusive value="0"/>
                            <xsd:maxInclusive value="100"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
            </xsd:all>
            <xsd:attribute name="nickname" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

In order to avoid confusion,
we need to see some examples..

QUIZ TIME!

Is it well formed?

1

```
<Pokemon nickname="GreenRipper" species="BULBASAUR" dex="1">
    <types type="GRASS" type="POISON"></types>
</Pokemon>
```

2

```
<Team>
    <Pokemon>
        <species name="BULBASAUR" dex="1" />
        <type>GRASS</type>
        <experience>64</experience>
        <moves>
            <attack>
                <name>RAZOR LEAF</name>
                <type>GRASS</type>
                <power>55</power>
            </attack>
            <effect />
        </moves>
    </pokemon>
</Team>
```

Is it well formed?

3

```
<T3am>
    <Poké-mon>
        BULBASAUR
    </Poké-mon>
</T3am>
```

4

```
<Team>
    <Pokémon>
        <!-- BULBASAUR DATA -->
    </Pokémon>
    <Pokémon>
        <!-- PIKACHU DATA -->
    </Pokémon>
</Team>

<Team>
    <Pokémon>
        <!-- CHARMANDER DATA -->
    </Pokémon>
    <Pokémon>
        <!-- SQUIRTLE DATA -->
    </Pokémon>
</Team>
```

5

```
<team>
    <pokemon>
    <bulbasaur>
    </pokemon>
    </bulbasaur>
</team>
```

Is it valid?

XSD

XML

```
<effect xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="pokemonNamespace"
xsi:schemaLocation="pokemonNamespace pokemon.xsd">
    <id>5</id>
    <type>NORMAL</type>
    <name>CONFUSION</name>
</effect>
```

```
<xsd:element name="effect" type="pkm:effect" />

<!--Pokémon effect attack-->
<xsd:complexType name="effect">
    <xsd:sequence>
        <!--Id of the effect-->
        <xsd:element name="id" type="xsd:string"/>

        <!--Name of the effect-->
        <xsd:element name="name" type="xsd:string"/>

        <!--Type of the effect (e.g. WATER, ELECTRIC,...)-->
        <xsd:element name="type" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

Declaration in XML are going to be omitted in these examples, xsd namespace definition as well: don't count them as errors

Is it valid?

XML

```
<gender xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="pokemonNamespace"
xsi:schemaLocation="pokemonNamespace pokemon.xsd">
    <female>1</female>
    <female>1</female>
</gender>
```

XSD

```
<xsd:element name="gender" type="pkm:gender" />

<!--Pokémon gender-->
<xsd:complexType name="gender">
    <xsd:choice>
        <xsd:element name="male" type="xsd:boolean" minOccurs="1"/>
        <xsd:element name="female" type="xsd:boolean" minOccurs="2"/>
    </xsd:choice>
</xsd:complexType>
```

Declaration in XML are going to be omitted in these examples, xsd namespace definition as well: don't count them as errors

Is it valid?

XSD

XML

```
<gender xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="pokemonNamespace"
xsi:schemaLocation="pokemonNamespace pokemon.xsd">
    <male>1</male>
    <male>false</male>
</gender>
```

```
<xsd:element name="gender" type="pkm:gender" />

<!--Pokémon gender-->
<xsd:complexType name="gender">
    <xsd:choice minOccurs="0">
        <xsd:element name="male" type="xsd:boolean" maxOccurs="2"/>
        <xsd:element name="female" type="xsd:boolean" maxOccurs="2"/>
    </xsd:choice>
</xsd:complexType>
```

Is it valid?

XSD

XML

```
<trainer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="pokemonNamespace"
xsi:schemaLocation="pokemonNamespace pokemon.xsd">
    <name>ASH</name>
    <exp>1</exp>
    <accessories>
        <pokeball>MEGABALL</pokeball>
        <tool>RADAR</tool>
        <pokeball>ULTRABALL</pokeball>
    </accessories>
</trainer>
```

```
<xsd:element name="trainer">
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="exp" type="xsd:integer"/>
            <xsd:element name="accessories">
                <xsd:complexType>
                    <xsd:choice minOccurs="0" maxOccurs="3">
                        <xsd:element name="pokeball" type="xsd:string" />
                        <xsd:element name="tool" type="xsd:string" />
                    </xsd:choice>
                </xsd:complexType>
            </xsd:element>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
```

Declaration in XML are going to be omitted in these examples, xsd namespace definition as well: don't count them as errors

Is it valid?

XSD

XML

```
<trainer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="pokemonNamespace"
xsi:schemaLocation="pokemonNamespace pokemon.xsd">
    <name>ASH</name>
    <exp>1</exp>
    <accessories></accessories>
</trainer>
```

```
<xsd:element name="trainer">
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="exp" type="xsd:integer"/>
            <xsd:element name="accessories">
                <xsd:complexType>
                    <xsd:choice minOccurs="0" maxOccurs="2">
                        <xsd:element name="pokeball" type="xsd:string" />
                        <xsd:element name="tool" type="xsd:string" minOccurs="1" />
                    </xsd:choice>
                </xsd:complexType>
            </xsd:element>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
```

Declaration in XML are going to be omitted in these examples, xsd namespace definition as well: don't count them as errors

XML: is it well formed ? / is it valid?

Go into the folder `Documents/json-xml-SDElab/xml/`.
The data of your team is going to be in the file `xmlInstances/myTeam.xml` *while the xsd definition is in* `xsdDefinitions/pokemonTeam.xsd`.

In order to call the <u>validation</u> script, you should open a terminal, move to the folder mentioned above and write: `node validate.js xmlInstances/myTeam.xml xsdDefinitions/pokemonTeam.xsd`

XPath

Addressing elements and attributes of an XML document

"Path-like" syntax to navigate through the "XML tree"

Select from the current node

Filter by attribute

`/tagname[@Attribute='Value']`

Name of targeted tag

Location Path
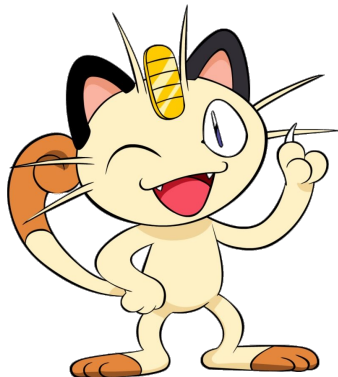
XPath - Syntax

location path

`/`**`child`**`::`**`Team`**`/`**`child`**`::`**`Pokemon`**`[`**`species=’BULBASAUR’`**`]`

location step

| Axis | Node test | Predicate |

Note: there is also an abbreviated syntax: e.g. `/Team/Pokemon[species=’BULBASAUR’]`

XPath - Tree navigation

`/Team/`**`Pokemon`**`[species='BULBASAUR']`
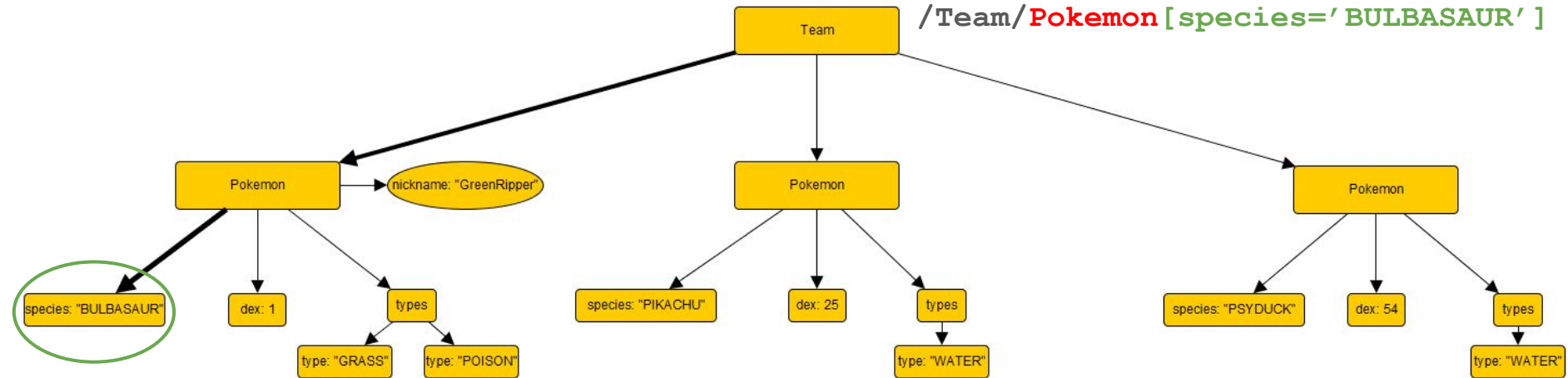
```
Element='<Pokemon xmlns="pokemonNamespace" nickname="GreenRipper">
            <species>BULBASAUR</species>
            <dex>1</dex>
            <types>
                    <type>GRASS</type>
                    <type>POISON</type>
            </types>
        </Pokemon>'
```

XPath

SQL

`/Team/Pokemon[@nickname='GreenRipper']`

```
SELECT *
FROM Pokemon P
WHERE nickname = "GreenRipper" AND
        P.team IN (SELECT id
                    FROM Team T
                    WHERE T.id = P.team)
```

Extracts all the pokemons that have GreenRipper as a nickname

XPath - Examples

`/Team/Pokemon`

Selects **the Pokemons**, which in the XML are **children** elements of the **root element Team**

`//moves/attack`

Selects **the attacks**, which in the XML are **children** elements of the **root or other deeper descendents** that in this case are **elements of type "moves"**

`//moves/attack[last()]`

Works just like above, then selects the **last attack** of each pokemon

`//Pokemon[@nickname='TopoGigio']`

Selects **the Pokemons**, which in the XML are **children** elements of the **root or other deeper descendents that has the nickname "TopoGigio"**

QUIZ TIME!

XPath - Quiz

`/Team/Pokemon[dex>=2]`

XPath - Quiz

`//moves/attack/*[last()]`

XPath - Exercises

Go into the folder `Documents/json-xml-SDElab/xml/`*. The data of
your team is going to be in the file* `xmlInstances/myTeam.xml`
In order to call an xPath <u>**query**</u> *on it, you should open a terminal, move
to the folder mentioned above and write:*
`node xPath.js xmlInstances/myTeam.xml` "<u>**query**</u>"

<mark>Note: YOU MUST USE APPROPRIATE NAMESPACE(s): e.g. //pkm:moves</mark>

1. Find all the pokemons in your team that have an experience level lower than 50
2. Find all the bulbasaur pokemons that have an attack with a power level higher than 40
3. Find the last pokemon in your team with sleep powder as an effect listed in the moves

JSON - JavaScript Object Notation[1]

Standard data representation and data **exchange** format

Easily generated and **readable** for both humans and machines

Data type defined in forms of **array**, **maps** (key value pairs) or both

Platform independent

(UTF-8 compliant)

Uses conventions present in other programming languages

Starting from primitive types (e.g. number, string, boolean)

1: Related/close to a JS object, but not completely the same (e.g. quotes, function type in js)

JSON - Basic concepts

```json
{
  "Team":[
    {
      "@nickname": "GreenRipper",
      "species": "BULBASAUR",
      "dex": "4",
      "types": [
        "GRASS",
        "POISON"
      ],
      "experience": "64",
      "moves": [
        {
          "name": "RAZOR LEAF",
          "type": "GRASS",
          "power": "55"
        },
        {
          "name": "SLEEP POWDER",
          "type": "GRASS"
        }
      ]
    }
  ]
}
```

- **Property Name**
  (always between double quotes)

- **Property Value**
  (could be a number, a string, a boolean, an object or an array)

- **Array**

  (a list composed of all sort of elements: primitive or composite)

- **Object**

  (map of *"key": value* properties)

Is JSON better than XML?
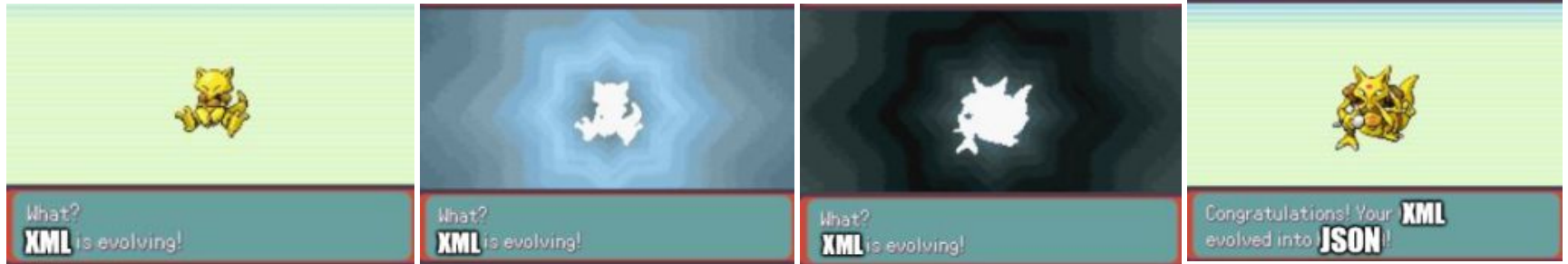
every evolution comes with its pros & cons...

{ JSON }

<XML />

{ JSON }	**<XML />**
Simple syntax (less verbose)	*Established <u>validation</u> techniques (standard)*
Easiness to use: time saving	*Strictness: more suitable to safety critical use cases*
Faster to transmit	There are **MORE STANDARDs**

serialization/deserialization faster in …
(depends)

From XML to JSON

Can be done in both ways using, for example,
the **xml2json** library in js:

- Easy to use (just call the function)
- Customizable
- Conversion is lossy

```
parser.toJson(xml, options);
```

```
parser.toXml(json);
```

Good results are not always guaranteed!

From XML to JSON

```xml
<Pokemon nickname="GreenRipper">
    <species>BULBASAUR</species>
    <dex>1</dex>
    <types>
        <type>GRASS</type>
        <type>POISON</type>
    </types>
</Pokemon>
```

Parser.toJson()

```json
{
    "Pokemon": {
        "nickname": "GreenRipper",
        "species": "BULBASAUR",
        "dex": 1,
        "types": [
            "GRASS",
            "POISON"
        ]
    }
}
```

Could be useful, but be aware you're going to **lose some information**
(e.g. XML attributes vs XML simple elements)

From JSON to XML

```json
{
  "Pokemon": {
    "nickname": "GreenRipper",
    "species": "BULBASAUR",
    "dex": 1,
    "types": [
      "GRASS",
      "POISON"
    ]
  }
}
```

Parser.toXml()

```xml
<Pokemon          nickname="GreenRipper"
                  species="BULBASAUR" dex="1"
                  types="GRASS" types="POISON">
</Pokemon>
```

Every time there is an array of simple elements in JSON,
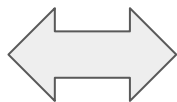the converted XML will be **NOT WELL-FORMED!**

JSON ⟷ XML

When can I make
"safe" conversions?

{ JSON }   <XML />

There aren't any arrays   ⟷   Leaf elements have just
                                    attributes and no content

JSON ⟺ XML **(don't abuse!)**

"Bulbasaur.xml"

```xml
<Pokemon nickname="GreenRipper">
    <species>BULBASAUR</species>
    <dex>1</dex>
    <types>
        <type>GRASS</type>
        <type>POISON</type>
    </types>
</Pokemon>
```
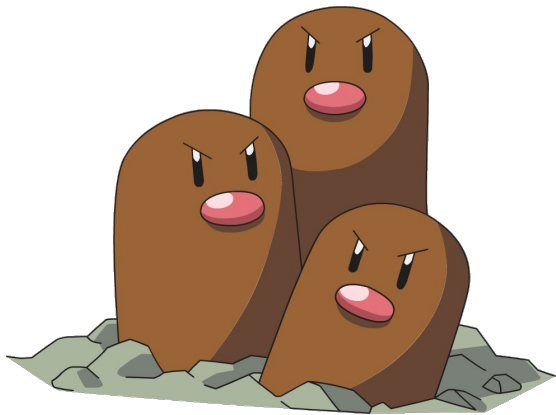
Parser.toJson();
Parser.toXml();

```xml
<Pokemon nickname="GreenRipper" species="BULBASAUR" dex="1">
    <types type="GRASS" type="POISON"></types>
</Pokemon>
```

XML not Well Formed!

"Bulbasaur.json"

```json
{
    "Pokemon":{
        "nickname": "GreenRipper",
        "species": "BULBASAUR",
        "dex": 1,
        "types": ["GRASS", "POISON"]
    }
}
```

Parser.toXml();
Parser.toJson();

Error while parsing XML!
Duplicated attribute "types"

JSON ⟷ XML **(don't abuse!)**

A few conversions later...

JSON - XML converter

Go into the folder `Documents/json-xml-SDElab/json2xmlExamples/.`
In order to call the **converter** *script, you should open a terminal, move to the folder mentioned above and write:*

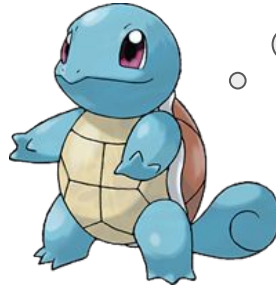`node xml2json.js <xmlInstance.xml>`   (from xml to json)
`node json2xml.js <jsonInstance.json>`   (from json to xml)

JSON - Properties

Checking for appropriate nesting

In js, I can just call `JSON.parse()`

WELL-FORMED ✔

JSON - Properties

There are methods to do it, but **NO** actual **STANDARDS**

We can use JSON schema

VALIDATION

JSON Schema

Provides a huge variety of **constraints**

Makes **combining schemas simpler**, implying smaller level of effort

Some **options** from XSD are **missing**

dependencies, max/minItems, requires, ...

anyOf, allOf, oneOf

Doesn't provide sequence check

JSON Schema

- **Constraints**
  (# of items, required fields, additional properties,..)

- Schema declaration
  (draft version of JSON schema)

- **Reference to the definition**
  (modularity and reusability)

- Basic types
  (integer, number, boolean, string, object, array)

```json
{
  "$id": "pokemonTeamJSONSchema",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "description": "A representation of a pokemon team",
  "type": "array",
  "items": { "$ref": "#/definitions/Pokemon" },
  "maxItems": 6,

  "definitions": {
    "Pokemon": {
      "type": "object",
      "properties": {
        "@nickname":{ "type": "string" },
        //other properties[...]
        "experience":{ "type": "integer", "minimum": 1, "maximum": 100 },
        "moves":{
          "type" : "array",
          "items":
            { "anyOf":
              [   { "$ref": "#/definitions/attack" },
                  { "$ref": "#/definitions/effect" }]
            },
          "minItems": 0, "maxItems": 4
        }
      },// other definitions[...]
      "required": ["experience", "moves", // other required properties*/]
    }
  }
}
```

Again...

we need to see some examples

QUIZ TIME!

Is it well formed?

(1)

```
{
    "Team":
    [
        {
            "nickname": 'GreenRipper',
            "species": 'BULBASAUR',
            "dex": 1,
            "types": ['GRASS', 'POISON'],
            "experience": 64,
            "moves":
                [
                    {
                        "name": 'RAZOR LEAF',
                        "type": 'GRASS',
                        "power": 55
                    }
                ]
        }
    ]
}
```

Is it well formed?

(2)

```json
{
    "Team":
    {
        {
            "nickname": "GreenRipper",
            "species": "BULBASAUR",
            "dex": 1,
            "types": ["GRASS", "POISON"],
            "experience": 64,
            "moves":
                [
                    {
                        "name": "RAZOR LEAF",
                        "type": "GRASS",
                        "power": 55
                    }
                ]
        }
    }
}
```

1

Is it valid?

SCHEMA

```
{
    "$id": "pokemonJSONSchema",
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "properties": {
        "nickname":{"type": "string"},
        "species": {"type": "string"},
        "types":{
            "type": "array",
            "items": {"type": "string"},
            "minItems": 1,"maxItems": 2
        }
    },
    "required": ["nickname", "species", "types"],
    "dependencies": {},
    "definitions": {}
}
```

JSON

```
{
    "nickname": "Bulby",
    "species": "BULBASAUR",
    "types": ["GRASS", "POISON"],
    "dex": 5
}
```

Is it valid?

SCHEMA

JSON

```json
[
  {
    "nickname": "GreenRipper",
    "species": "BULBASAUR",
    "types": ["GRASS", "POISON"]
  }
]
```

```json
{
  "$id": "pokemonJSONSchema",
  "$schema": "http://json-schema.org/draft-07/schema#",

  "Pokemon":{
    "type": "object",
    "properties": {
      "nickname":{"type": "string"},
      "species": {"type": "string"},
      "dex": {"type": "integer"},
      "types":{
        "type": "array",
        "items": {"type": "string"},
        "minItems": 1,"maxItems": 2
      }
    },
    "required": ["species", "dex", "types"]
  },
  "definitions": {}
}
```

3

Is it valid?

JSON

```
{
  "nickname": "GreenRipper",
  "species": "BULBASAUR",
  "types": ["GRASS", "POISON"],
  "dex": 1,
  "accessory": "Peculiar Spoon"
}
```

SCHEMA

```
{
  "$id": "pokemonJSONSchema",
  "$schema": "http://json-schema.org/draft-07/schema#",

  "type": "object",
  "properties": {
    "nickname":{ "type": "string" },
    "species": { "type": "string" },
    "dex": { "type": "integer" },
    "types":{
      "type": "array",
        "items": {"type": "string"},
        "minItems": 1,
        "maxItems": 2
    }
  },
  "required": ["species", "dex", "types", "accessory"]
}
```

JSON: is it well formed ? / is it valid?

Go into the folder `Documents/json-xml-SDElab/json/.`
The data of your team is going to be in the file `jsonInstances/myTeam.json` *while the json schema definition is in* `jsonDefinitions/pokemonTeam.json.`

In order to call the <u>validation</u> script, you should open a terminal, move to the folder mentioned above and write: `node validate.js jsonInstances/myTeam.json jsonDefinitions/pokemonTeam.json`
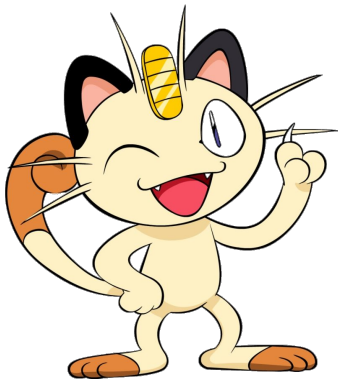
JSONata - Introduction

Query language for JSON

Similar to what XPath does for an XML document

Select from the current node

Filter by property value

`.property[property=value]`

Name of targeted property

Why JSONata?

There are different libraries, with more "normal" names, like **JSON Pointer** or **JSONPath**.

You could think they work, well.. **<u>DON'T</u>** use those libraries.

They are terrible, like Bidoof.

JSONata - Syntax

`$.Team[species='BULBASAUR']`

Root	Property	Expression

Simply extracting all the pokemons in the team which are of the BULBASAUR species

JSONata - Examples

`$.Team`

Selects the Pokemons, which in the JSON are elements of the array **Team** which is a child element of the **root**

`$.**.moves`

Selects **all the moves**, which in the JSON are **children** elements of the **root or other deeper descendents**

`$.Team[dex>4].moves[-1]`

Selects the **last move** of **all the pokemons** that **have a dex level greater than 4**

`$.Team['GRASS' in types]`

Selects the Pokemons, which are elements of the array **Team** (which is a child element of the **root**), **that have 'GRASS' as an element of the array types**

QUIZ TIME!

JSONata - Quiz

```
$.Team[moves[type="GRASS"][power<=110]]
```

JSONata - Quiz

```
$.Team.moves[-2]
```

JSONata - Exercises

Go into the folder `Documents/json-xml-SDElab/json/`. *The data of your team is going to be in the file* `myTeam.json`
In order to call a JSONata <u>query</u> on it, you should open a terminal, move to the folder mentioned above and write:
`node jsonata.js myTeam.json` <u>query</u>

1. Find all the pokemons in your team that have an experience level lower than 50
2. Find all the bulbasaur pokemons that have a move with a power level higher than 40
3. Find the last pokemon in your team with sleep powder listed in the moves

References

XML   [https://www.w3.org/XML/Core/#Publications]

XSD   [https://www.w3.org/TR/xmlschema11-1/]

XPath   [https://www.w3.org/TR/2017/REC-xpath-31-20170321/]

JSON   [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON]

JSON Schema   [https://json-schema.org/understanding-json-schema/]

JSONata   [http://docs.jsonata.org/overview]

We hope our cheat sheets will be useful too!

Thanks for your attention!
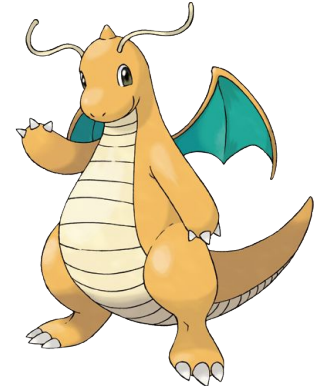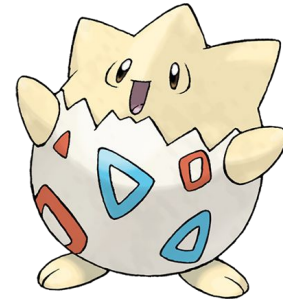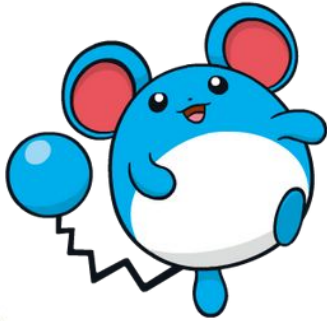
Hope you've had fun

(well... if by any chance you've learnt something, it's not bad either)